

Borland[®]



Desvendando o Caminho das Pedras

Fernando Anselmo

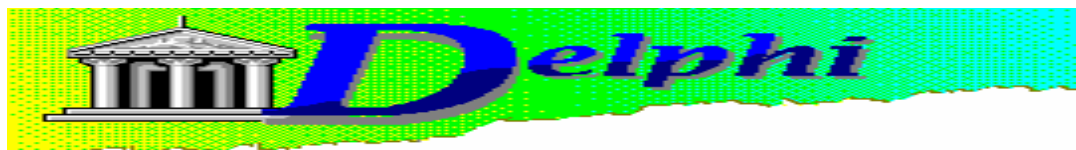
Dez 1995 - Mai 1997

Desvendando o Caminho das Pedras

Copyright © 1995-97 Fernando Antonio F. Anselmo.

Todos os nomes dos produtos citados são marcas registradas da Borland International, Inc.
Outros produtos citados são marcas registradas no respectivo cabeçalho

As várias **Marcas Registradas** que aparecem no decorrer deste livro. Mais do que simplesmente listar esses nomes e informar quem possui seus direitos de exploração, ou ainda imprimir o logotipo das mesmas, o autor declara estar utilizando tais nomes apenas para fins editoriais, em benefício exclusivo do dono da marca registrada, sem intenção de infringir as regras de sua utilização.



SUMÁRIO

Desvendando o Caminho das Pedras	i
----------------------------------	---

INTRODUÇÃO

<i>Delphi</i> , como Solução para Desenvolvedores	7
---	---

CAPÍTULO I

9

Conceito de Programação Orientada a Objeto

9

Orientação a Objeto	9
Object Pascal	10
Símbolos Especiais	10
Palavras Reservadas	11
Números	12
Constantes	12
Expressões	12
Identificadores	13
Declarações	14
Blocos de Procedimentos ou Funções	16
Características de Objetos	17
Programando com objetos <i>Delphi</i>	18
Renomeando os objetos e os componentes	20

CAPÍTULO II

21

Conhecendo o Delphi

21

Elementos Visíveis	21
Form	22
Code Editor	23
Component Palette	23
Object Inspector	24
SpeedBar	24
Elementos não Visíveis	24
Project Manager	25
Menu Designer	25
Fields Editor	26
Repositório de Objetos	27
Estrutura de Aplicações com o <i>Delphi 2.0</i>	28
Implementação efetiva	29
Objetos Data Module	29
Dicionário de Dados Escalável	30
Herdando os Formulários	31
Ferramentas Auxiliares de SQL	32
Monitor SQL	32
SQL Explorer	33
InterBase NT - Banco de Dados Relacional	34

CAPÍTULO III

35

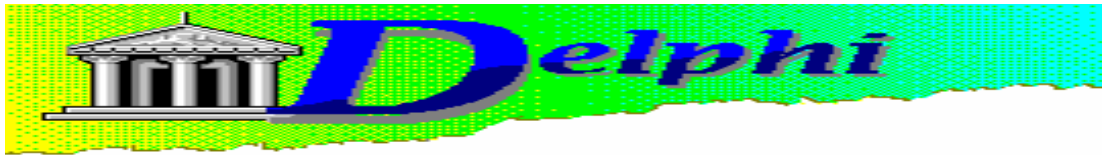
Projeto Piloto

35

Criando o Modelo Relacional	36
Trabalhando com DataBase Engine Configuration	37
Criando o Alias	37
Trabalhando com DataBase DeskTop	38
Criando o Banco de Dados via Estrutura	38
Criando os Relacionamentos via Estrutura	40
Criando o Banco de Dados via SQL	40
Observações da utilização do SQL com o dBase	41
 CAPÍTULO IV	 43
Trabalhando com o Menu	43
Metendo a Mão na Massa	43
Criando a janela do menu	43
Inserindo os Códigos Iniciais	46
Iniciando os comandos do Menu	47
Colocando os comandos para o Auxílio	48
Criando a janela “Sobre o Sistema”	50
Criando e alterando os objetos	50
Associando o form “Sobre o Sistema” ao menu	51
Criando a janela <i>Splash</i>	52
Criando o acesso a Base de Dados	54
 CAPÍTULO V	 57
Janela para as Tabelas	57
Reabrindo o seu Projeto	57
Alterando as Janelas Criadas	58
DataModules ?	58
Modificando as Tabelas e as Ligações	58
Alterando os campos da tabela	59
Codificando o DataModule	60
Controlando a duplicação dos Campos-Chave	61
Finalizando o DataModule	62
Alterando o Formulário	63
Modificando os Labels e Campos	64
Objeto DBNavigator	65
Modificando os Paineis	65
Modificando a Janela	66
Criando o terceiro Painel	66
Programando no formulário	68
Preservando as áreas de Memória	68
Criando Funções Globais	71
Alterando o Menu para receber o formulário	74
 CAPÍTULO VI	 77
Trabalhando com janela Pai X Filha	77
Criando a janela automaticamente	77
Sobre os DataModules	79
Trabalhando com as Tabelas	79
Trabalhando com os Campos	81
Controlando o DataModule	82
Contadores	83
Validando os Campos	84
Alterando a Janela Criada	87

Organizando os Panels	87
Modificando os campos e Labels	88
Organizando os Panels	88
Modificando a Janela	89
Trabalhando com Grid's	91
Finalmente, a programação	93
Consulta	94
Trabalhando com a área de Transferência	96
Utilizando o objeto OpenDialog	96
Criando o formulário para o cadastro das músicas	97
Criando novos Procedimentos Globais	100
Alterando o Menu para receber o formulário	102
CAPÍTULO VII	106
Trabalhando com consultas	106
Criando consultas para tabelas	106
Trabalhando com Grid's	106
Programando no formulário	109
Enviando e recebendo variáveis	110
Alterando o formulário fCateg	110
Alterando o formulário fBasico	111
Criando consultas para o cadastro	114
Consultas SQL	114
Realizando Consultas com Filtros	114
Programando o formulário	118
Criando o formulário Gerente do Filtro	125
Programando o formulário	128
Editando os registros	129
CAPÍTULO VIII	132
Relatórios	132
Trabalhando com o ReportSmith	132
Criando relatório com o ReportSmith	133
Organizando os campos do relatório	135
Associando o relatório ao aplicativo	135
Programando o formulário	137
Imprimindo através do Formulário	138
Criando o Código	140
Trabalhando com o QuickReport	141
CAPÍTULO IX	146
Multimídia	146
O que é multimídia ?	146
Delphi and Multimedia	147
Objeto TMediaPlayer	147
Colocando as propriedade em modo Runtime	148
Pesquisando variáveis em modo RunTime	149
Inserindo o multimídia para o Sistema	152
Desenvolvimento do CD Player	152
CAPÍTULO X	158

Novos Componentes	158
Criando Componentes	158
A Classe TComponent	159
Um Componente Simples	159
Adicionando o Componente a Palheta	160
Criando Propriedades	161
Métodos de Acesso	161
Criando novos tipos	162
Pensando em Objetos	163
Construindo um Objeto	163
Finalmente	167
 APÊNDICE A	 168
Documentação	168
Hardware/Software requeridos	168
 APÊNDICE B	 169
Conversão de Campos	169
Tipos de Dados para o InterBase	170
 APÊNDICE C	 171
Aplicação rápida com o Objeto Query	171
 APÊNDICE D	 173
Imprimindo um Formulário	173
 APÊNDICE E	 175
Trabalhando com Máscaras	175
 APÊNDICE F	 177
Trabalhando com Importação e Exportação	177
 APÊNDICE G	 180
Doze melhores dicas para o Delphi	180



Introdução

Bem-vindo ao *Delphi*, o mais novo produto de alta performance da *Borland*. *Delphi* é um produto único em sua categoria combinando códigos totalmente compiláveis, ferramentas visuais e tecnologia para a composição de bases de dados escaláveis, possui facilidades para um rápido desenvolvimento em plataforma *Windows*® e aplicações *Client/Server*.

Este trabalho será seu guia para uma rápida aprendizagem no desenvolvimento de sistemas que gerencie bancos de dados. O *Delphi* é encontrado em dois produtos:

- ***Delphi Client/Server***, de alta performance e facilidade para o desenvolvimento de aplicações e suporte a bancos de dados do tipo Cliente/Servidor.
- ***Delphi Desktop***, de alta performance e facilidade para o desenvolvimento de aplicações e suporte a bancos de dados locais, permitindo total portabilidade à versão Client/Server.

Apresento-lhes a seguir algumas informações detalhadas para um perfeito desenvolvimento visual, sendo que ao final de cada capítulo prático é exibido o código fonte completo seguido de um resumo dos principais comandos mostrados. Ao final deste estudo você encontrará apêndices que lhe ajudarão a resolver pequenos problemas do dia-a-dia.

Delphi, como Solução para Desenvolvedores

Muitas vezes nos perguntamos, e somos questionados, no porque de adotar o *Delphi* como a linguagem para o desenvolvimento de sistemas ? Inicialmente, é necessário conhecer que o *Delphi* oferece um rápido caminho para o desenvolvimento de aplicações nos ambientes:

- *Windows*®, *Windows 95*® e *Windows NT*®;
- Bancos de dados do tipo Cliente/Servidor: *Oracle*®, *Informix*®, *InterBase*, *SyBase*® e *Microsoft SQL Server*®;
- Alta performance, em sistemas críticos;
- Base de Dados locais e aplicações do tipo network;
- Ambiente gráfico, visual e multimídia.

Mas o que é possível fazer com ele ? É possível criar, dentre outros, os seguintes tipos de aplicações em *Delphi* :

- Usá-lo como a linguagem de desenvolvimento para bancos do tipo Cliente/Servidor;
- Ambiente heterogêneo para captura e envio de informações em diversos tipos de arquivos de dados;

- Um pacote corporativo de aplicações inteligentes e interpretadores de dados. Incorporando DLL's e EXE's externos;
- Pacotes multimídia com desenho e animação;
- Genéricos utilitários do Windows[®];
- Criação de bibliotecas (DLL) para leitura por outras aplicações.

Mas porque arriscar em um ambiente novo quando existe no mercado linguagens mais difundidas ? No mundo inteiro *Delphi* foi testado, e em 15 meses de vida produziu os seguintes resultados:

- *Delphi* está sendo utilizado no momento por mais de 1.500 lugares incluindo as maiores corporações, consultores e organizações de treinamento;
- Eleito pela Byte Magazines como Best of Comdex Award;
- Vários livros escritos;
- Grupos de discussão e periódicos com dicas de desenvolvimento na WorldWibe (Consulte às listas da InterNet através da palavra **DELPHI**);
- Dezenas de bibliotecas e ferramentas para o suporte em *Delphi*;
- Dezenas de artigos em publicações do mundo inteiro, tais como *PC Week*, *InfoWorld*, *Computer Reseller News*, *PC Magazine*, *Windows Sources* e muitas outras.

Por tudo aqui exposto fica claro que este no produto demonstra uma inovação para uma criação em alta performance de aplicações. Todos os recursos que você precisará para o desenvolvimento de seus produtos estão agora disponíveis.

Feliz desenvolvimento.

Fernando Antonio F. Anselmo

U'Sempre que você localizar este símbolo significa que existe uma nota que lhe ajudará em caso de dúvida.

Capítulo I

Conceito de Programação Orientada a Objeto

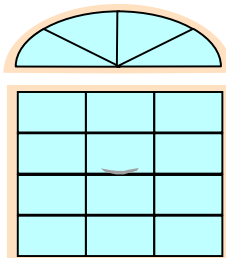
Para compreendermos melhor a novo ambiente de desenvolvimento da *Borland* o *Delphi* é necessário que você, aprenda e, tenha em mente os conceitos de POO (Programação Orientada a Objetos), não confunda os conceitos com POE (Programação Orientada a Eventos) muito difundido com o *Access 2.0*® (um ambiente baseado em Objetos), mas ao longo deste capítulo você vai notar as sensíveis diferenças que existem entre esses dois conceitos.

A POO e a POE são facilmente confundidas, mas lembre-se a POO contém a POE mas a POE não contém a POO, um objeto pode existir mesmo que não exista nenhum evento associado a ele, mas um evento não pode existir se não houver um objeto a ele associado. Outra característica que pode causar confusão são ambientes Orientados a Objetos e ambientes Baseados em Objetos. Em ambiente Orientado a Objetos consegue-se criar e manipular objetos enquanto que o Baseado em Objetos não é possível a criação de objetos apenas a sua manipulação.

A POO é um conceito desenvolvido para facilitar o uso de códigos de desenvolvimento em interfaces gráficas. Sendo a *Borland*, uma das primeiras a entrar neste novo conceito, possui suas principais linguagens de programação (tais como *Object Pascal* e *C++*), totalmente voltadas para este tipo de programação. A POO atraiu muitos adeptos principalmente pelo pouco uso de código que o projeto (diferente de sistema) carrega no programa fonte, ao contrário das linguagens mais antigas como o *Clipper'87*® muito utilizado no final da década de 90 e início da década de 90. O resultado desta “limpeza” no código resulta que a manutenção do projeto torna-se muito mais simples.

Orientação a Objeto

Antes de começarmos a falar realmente de linguagem orientada a objetos e necessário que você possua os conceitos básicos da orientação a objetos, são eles:



Objeto - é qualquer estrutura modular que faz parte de um produto. Uma janela por exemplo, é um objeto de uma casa, de um carro ou de um software com interface gráfica para o usuário.

Atributos - São as características do objeto, como cor e tamanho, a janela, por exemplo, tem atributos como o modelo, tamanho, abertura simples ou dupla, entre outros.

Encapsulação - é um mecanismo interno do objeto “escondido” do usuário. Uma pessoa pode abrir uma janela girando a tranca sem precisar saber o que há dentro dela.

Ação - é a operação efetuada pelo objeto. Todas as janelas, por exemplo, controlam a iluminação e temperatura ambiente, dependendo do seu design.

Herança - um objeto novo nem sempre é criado do zero. Ele pode “herdar” atributos e ações de outros já existentes. Um basculante herda atributos das janelas e das persianas.

Polimorfismo - é a capacidade de objetos diferentes reagirem segundo a sua função a uma ordem padrão. O comando “abre”, por exemplo, faz um objeto entrar em ação, seja ele uma janela, uma porta ou uma tampa de garrafa.

Ligação - é quando um objeto conecta a sua ação a outro. Um sensor de claridade, por exemplo, ativa o acendimento automático da iluminação de rua.

Embutimento - Permite a um objeto incorporar funções de outros, como um liquidificador que mói carne com a mudança do tipo da lâmina.

Object Pascal

Object Pascal é uma linguagem Orientada a Objetos não pura mas híbrida por possuir características de programação não só visual mas também escrita, para os programadores que já conhecem técnicas de estruturas de programação, com o *C*, *Basic*, *Pascal* ou *xBASE* entre outras linguagens a *Object Pascal* providência uma migração de forma natural oferecendo um produto de maior complexibilidade. *Object Pascal* força a você executar passos lógicos isto torna mais fácil o desenvolvimento no ambiente *Windows* de aplicações livres ou que utilizam banco de dados do tipo *Cliente/Servidor*, trabalha com o uso de ponteiros para a alocação de memória e todo o poder de um código totalmente compilável. Além disso possibilita a criação e reutilização (vantagem de re-uso tão sonhado com a **Orientação a Objetos**) de objetos e bibliotecas dinâmicas (*Dynamic Link Libraries* - DLL).

[AL1] Comentário:

Object Pascal contém todo o conceito da orientação a objetos incluindo encapsulamento, herança e polimorfismo. Algumas extensões foram incluídas para facilitar o uso tais como conceitos de propriedades, particulares e públicas, e tipos de informações em modo run-time, manuseamento de exceções, e referências de classes. O resultado de toda esta junção faz com que *Object Pascal* consiga suportar as facilidades de um baixo nível de programação, tais como:

- Controle e acesso das subclasses do *Windows* (API);
- Passar por cima das mensagens de loop do *Windows*;
- Mensagens semelhantes as do *Windows*;
- Código puro da linguagem *Assembler*.

Como deu para perceber a base de toda a programação *Delphi* é a linguagem *Object Pascal*, então neste capítulo trataremos exclusivamente deste tipo de programação.

Símbolos Especiais

A *Object Pascal* aceita os seguintes caracteres ASCII:

- ✓ Letras - do Alfabeto Inglês: **A** até **Z** e **a** até **z**.
- ✓ Dígitos - Decimal: **0** até **9** e Hexadecimal: **0** até **9** e **A** até **F** (ou **a** até **f**)
- ✓ Brancos - Espaço (**ASCII 32**) e todos os caracteres de controle **ASCII** (**ASCII 0** até **ASCII 31**), incluindo final de linha e Enter (**ASCII 13**).
- ✓ Especiais - Caracteres: + - * / = < > [] . , () : ; ^ @ { } \$ #
- ✓ Símbolos - Caracteres: <= >= := .. (* *) (.) //

U O colchetes esquerdo ([) e equivalente ao (. e o colchetes direito (]) e equivalente a .). A chave esquerda ({) e equivalente ao (* e a chave direita (}) e equivalente a *).

Palavras Reservadas

A **Object Pascal** se utiliza das seguintes palavras reservadas, não podendo as mesmas serem utilizadas ou redefinidas:

And	Exports	Library	Set
Array	File	Mod	Shl
As	Finally	Nil	Shr
Asm	For	Not	String
Begin	Function	Object	Then
Case	Goto	Of	To
Class	If	On	Try
Const	Implementation	Or	Type
Constructor	In	Packed	Unit
Destructor	Inherited	Procedure	Until
Div	Initialization	Program	Uses
Do	Inline	Property	Var
Downto	Interface	Raise	While
Else	Is	Record	With
End	Label	Repeat	Xor
Except			

Uma outra lista a seguir, apresenta as diretivas que são utilizadas em contextos de identificação de objetos:

Absolute	Export	Name	Published
Abstract	External	Near	Read
Assembler	Far	Nodefault	Resident
At	Forward	Override	Stored
Cdecl	Index	Private	Virtual
Default	Interrupt	Protected	Write
Dynamic	Message	Public	

Números

É possível definir variáveis e constantes de tipos de *Inteiro* ou *Real* através de qualquer decimal ordinário (**0** a **9**), mas a *Object Pascal* também aceita a notação Hexadecimal utilizados com o prefixo dollar (**\$**) ou a notação científica (**E**).

Constantes

Uma constante é um identificador com valor(es) fixo(s). Um bloco de declarações constante possui a seguinte expressão:

[Declaração Constante] [Identificador] (=) [constante] (;)

A lista abaixo apresenta um conjunto de funções que podem ser utilizadas para a declaração das constantes:

Ab	Length	Ord	SizeOf
Chr	Lo	Pred	Succ
Hi	Low	Ptr	Swap
High	Odd	Round	Trunc

Alguns exemplos para a definição de Constantes:

```
const Min = 0;
Max = 100;
Centro = (Max - Min) div 2;
Beta = Chr(225);
NumLetras = Ord('Z') - Ord('A') + 1;
MensOla = 'Instrução inválida';
MensErro = ' Erro: ' + MensOla + ' ';
PosErr = 80 - Length(MensErro) div 2;
Ln10 = 2.302585092994045684;
Ln10R = 1 / Ln10;
DigNumericos = ['0'..'9'];
LetrasAlpha = ['A'..'Z', 'a'..'z'];
AlphaNum = LetrasAlpha + DigNumericos;
```

Expressões

As expressões em Object Pascal (como em qualquer linguagem) é formada por operadores e operandos; os operadores são divididos em quatro categorias básicas:

Únicos	@, Not
Multiplicativos	>, /, div, mod, and, shl, shr, as
Adicionais	+, -, or, xor
Relacionais	=, <>, <, >, <=, >=, in, is

As expressões obedecem as regras básicas de lógica para a precedência da execução das operações.

Identificadores

Identificadores podem ser constantes, tipos, variáveis, procedures, funções, unidades, programas e campos de registros.

Não existe limite de caracteres para o nome de um identificador mas apenas os 63 primeiros caracteres são significantes (não podendo ser idêntico ao nome das palavras reservadas). O nome de um identificador deve ser iniciado por Letras ou o carácter underscore (_). O resto é formado por Letras, Dígitos, carácter underscore (ASCII \$5F). Não é permitido a utilização de espaços para a formação do nome.

U'Exemplo de identificadores válidos: Form1, SysUtils.StrLen, Label1.Caption

with... do...;

Delimita um determinado bloco de declarações para um identificador específico evitando a declaração deste identificador. A sintaxe do comando é: **WITH {nome do identificador} DO {comandos};**. Ex:

```
begin
  { ... comandos iniciais ... }
  with form1 do
    begin
      Caption := 'Teste';           Equivalente a Form1.Caption
      BorderStyle := bsSizable;     Equivalente a Form1.BorderStyle
    end;
end;
```

array [...] of ...;

Define um conjunto de variáveis ou constantes de um mesmo tipo. A sintaxe do comando é: **array [{quantidade de ocorrências}] of {Tipo};**. Os arrays são controlados por três funções:

Função	Valor de Retorno
Low	Primeiro elemento
High	Aponta para o último elemento
SizeOf	Tamanho do array

Ex:

```
const
  t: array [1..50] of Char { Declara 50 elementos para o tipo Char }
var
  s : array[1..100] of Real { Declara 100 elementos para o tipo real }
ind: Integer;
begin
  for Ind := Low(s) to High(s) do s[Ind] := 0; { Zera os elementos do array S }
  if SizeOf(t) = 'C' then exit; { Se o último elemento do array T for 'C' sai do bloco }
  { ... outros comandos... }
end;
```

Declarações

Declarações descrevem ações de um algoritmo a serem executadas.

begin... end;

Prende um conjunto de declarações em um bloco de comandos determinado. A sintaxe do comando é:

BEGIN {comandos} END; Ex:

```
begin
  { ... comandos iniciais ... }
  begin
    { ... bloco 1 ... }
  end;
  begin
    { ... bloco 2 ... }
  end;
  { ... comandos finais ... }
end;
```

if... then... else...;

Esta expressão escolhe entre o resultado de uma condição booleana o caminho verdadeiro (then) ou falso (else). A sintaxe do comando é: **IF {condição} THEN {bloco de comandos} ELSE {bloco de comandos};** Ex:

```
begin
  { ... comandos iniciais ... }
  if x > 2 then
    { ... Bloco verdadeiro ... }
  else
    { ... Bloco falso ... };
end;
```

goto... ;

Transfere a execução de um programa para o ponto determinado pelo **Label**. A sintaxe do comando é:

GOTO {Label}; Ex:

```
label
primeiro;
begin
  { ... comandos iniciais ... }
  if x = 2 then
    goto primeiro;
  { ... outros comandos ... }
Primeiro:
  { ... comandos do Primeiro ... }
end;
```

case... of... else... end;

Consiste de uma lista de declarações que satisfaz a condição de um seletor de expressões, se nenhuma parte da lista satisfizer ao seletor executa os comandos do sub-comando **else**. Para o seletor serão válidos os tipos definidos, tipo Inteiros ou LongInt. A sintaxe do comando é: **CASE {seletor} OF {Expressão 1}: {comando da expressão 1}; {Expressão 2}: {comando da expressão 2}; {Expressão n}: {comando da expressão n} ELSE {comando}; end;** Ex:

```
begin
  { ... comandos iniciais ... }
  case x of
    1: { ... Bloco para x = 1 ... }
    2, 3: { ... Bloco para x = 2 ou X = 3... }
    4..6: { ... Bloco para 4 <= x <= 6 ... }
  else
    { ... Bloco para x < 1 ou x > 6 ... };
  end;
end;
```

repeat... until;

Repete um determinado bloco de declarações até a condição booleana do subcomando **until** ser satisfeita. A sintaxe do comando é: **REPEAT {comandos}; until {condição};** Ex:

```
begin
  { ... comandos iniciais ... }
  x := 0;
  repeat
    x := x + 1
  until (x = 2);
end;
```

for... to (downto)... do...;

Incrementa em 1 uma determinada variável inteira, repetindo um bloco de comandos, até que esta atinja o valor final do intervalo, o subcomando **downto** realiza o incremento reverso. A sintaxe do comando é: **FOR {variavel} := {valor inicial} to (downto) {valor final} do {bloco de comandos};** Ex:

```
begin
  { ... comandos iniciais ... }
  for i := 1 to 10 do           Executa o [comandos A] para i = 1,2,3,4,5,6,7,8,9 e 10
  { ... Comandos A ... }
  for s := 10 downto 1 do      Executa o [comandos B] para i = 10,9,8,7,6,5,4,3,2 e 1
  { ... Comandos B... }
end;
```

while... do...;

Repete um bloco de comandos enquanto que determinada condição booleana seja satisfeita. A sintaxe do comando é: **WHILE {condição} DO {bloco de comandos};** Ex:

```
begin
  { ... comandos iniciais ... }
  while i := 1 do              Repete o [Bloco de comandos] enquanto i = 1
```

```
{ ... Bloco de comandos ... }  
end;
```

break; ou continue...;

O comando **break** interrompe um bloco de repetição **for**, **while** ou **repeat** saindo do bloco. A sintaxe do comando é: **BREAK**; enquanto que o comando **continue** retorna a primeira instrução do bloco de repetição **for**, **while** ou **repeat**. A sintaxe do comando é: **CONTINUE**;. Ex:

```
begin  
  { ... comandos iniciais ... }  
  for i := 1 to 10 do  
    begin  
      if i = 8 then  
        break;           Salta para os [comandos C]  
      { ... comandos A... }  
      if i = 5 then  
        continue;       Retorna para o comando for pulando os [comandos B]  
      { ... comandos B ... }  
    end;  
  { ... comandos C ... }  
end;
```

Blocos de Procedimentos ou Funções

As procedures ou funções são declaradas na seção de tipos de declarações (abaixo do comando **type**) pertencendo ao objeto ou serem do tipo **public** (públicas - executadas por outras unidades) ou **private** (particulares - restritas a unidade local).

Procedure

procedure {cabeçalho}; var {declaração das variáveis}; {bloco de comandos};

O cabeçalho da procedure é composto pelo nome do procedimento e variáveis que serão recebidas (ou modificadas através da declaração **var**, ex: procedure teste(**var** x:string);).

```
procedure soma(a,b: integer);           Início enviando as variáveis A e B do tipo inteiro.  
var                                     Declaração de variáveis locais.  
  c: integer;  
begin                                  Corpo do procedimento.  
  c := a + b;  
end;
```

Function

function {cabeçalho} : {resultado}; var {declaração das variáveis}; {bloco de comandos};

As funções se diferem dos procedimentos pela obrigatoriedade do retorno de um resultado, podendo este resultado ser retornado pela declaração: **{nome da função} := valor** ou **result := valor**.

```
function soma(a,b: integer) : integer;   Início enviando as variáveis A e B do tipo inteiro.
```



```
begin                                     Corpo do procedimento.
  soma := a + b;                          ou result := a + b;
end;
```

¶ Junto com o *Delphi 2.0* vem o manual de **Object Pascal** em formato **.HLP**, caso a linguagem seja novidade para você aconselho que você dê uma boa olhada (o *Delphi 1.0* traz o mesmo manual, mas em formato **.PDF**), mas não se preocupe com o que foi explicado acima já está mais do que suficiente para uma boa inicialização com o *Delphi*.

Tudo o que vimos a cima é o que normalmente temos em outras linguagens comuns, mas o caracteriza realmente a linguagem Orientada em Objetos é o trabalho e a manipulação com os mesmos.

Características de Objetos

Mas afinal de contas, o que é um objeto ? Como foi dito anteriormente, um objeto é qualquer tipo de elemento, ou componente, que envolva dados e código dentro de um único pacote.

Uma vantagem de programar na POO e quanto a *Herança* dos objetos, este método faz com que seja possível um objeto 'Filho' poder herdar todas as características e conteúdos de um objeto 'Pai'. Tirando um pouco do *Pascal* da geladeira (a partir do *Pascal* versão 7.0 a *Borland* tornou possível a utilização simplificada de todo o conceito de POO) aqui vai um código completo de declaração de dois objetos, o primeiro chamado de TPai e o segundo de Tfilho:

```
TPai = object
  Nome: PChar;
  constructor Init (P: PChar);
  destructor Done;
  procedure MudaNome(P: PChar);
  procedure ShowName;
end;
```

```
TFilho = object(TPai)
  procedure MudaNome(P: PChar);
end;
```

O segundo objeto Tfilho herda do objeto TPai o ponteiro variável **Nome**, a **constructor Init**, o **destructor Done** e a **procedure ShowName**, apenas a **procedure MudaNome** terá o funcionamento como uma característica única para cada objeto. O *Delphi* possui inúmeros "pais" (classes de objetos) prontos para serem usados por você, tais como:

TForm: Centro das aplicações *Delphi*, utilizados na criação de janelas, caixas de diálogo entre outros.

TMenu: Responsável pela concepção de menus e menu popup.

TButtonControl: Simplifica o refinamento do controle da janela serve de base para os componentes como: Botões, Check Box e Radio Box.

Programando com objetos *Delphi*

Quando iniciado o *Delphi*, é criado automaticamente um novo projeto e um objeto formulário (derivado da classe **TForm**) para o suporte dos demais objetos. Explorando o Editor de Códigos (*Code Editor*) você poderá observar a declaração do novo objeto da classe *TForm* que foi produzido automaticamente com a criação do novo formulário. Examinando o conteúdo deste código criado para o objeto, teremos:

```
unit Unit1;                                Abertura de uma nova unidade

interface                                  Parâmetros do objetos

uses                                        Uso de outras unidades
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;

type
  TForm1 = class(TForm)                    A declaração do objeto inicia aqui
  private
    { Private declarations }
  public
    { Public declarations }
  end;                                      Aqui é o final da declaração

var
  Form1: TForm1;                           Atribui a variável Form1 as características do objeto TForm1

implementation                             Início da parte a ser implementada

{$R *.DFM}                                Diretiva de compilação que agrega o desenho da tela (em
                                           .DFM) com o mesmo nome da unidade

end.                                       Final da parte implementada
```

Um novo tipo de objeto *TForm1*, é declarado derivado da classe *TForm*, que também é um outro objeto. Relembre um objeto é um tipo de elemento capaz de guardar dados e código dentro de um único pacote. Até agora, o tipo *TForm1* não contém campos ou métodos, isso acontecerá com a adição de alguns componentes neste objeto.

Observando o código, notamos que existe uma variável declarada com o nome *Form1* para o novo tipo de objeto *TForm1*:

```
var
  Form1: TForm1;
```

Form1 é a chamada de instância ao tipo *TForm1*. Esta variável refere-se ao formulário em si, aonde será adicionado componentes e desenhado a interface entre o computador e o usuário que for operar o sistema. É sempre notado declarações de uma ou mais instâncias referidas ao tipo de objeto. Futuramente será mostrado o poder deste tipo de declarações quando falarmos sobre janela MDI (*Multiple Document Interface* - Interface de documento múltiplos) gerenciando várias “janelas filhas”, não permitindo que estas “janelas filhas” saiam do espaço criado pela “janela pai”.

Adicionando alguns componentes ao formulário, veremos como o *Delphi* completará a aplicação escrevendo automaticamente o código, e permitindo que ao final tornar-se-á possível a compilação (lembra-se do *Clipper*®, com **.EXE**), execução e distribuição da aplicação.

Em nosso formulário, colocaremos um botão que, em tempo de execução, ao ser dado um clique com o mouse sobre este objeto, o formulário mude sua cor. Aperte a tecla **F12** para retornar à visão do formulário e na palheta de objetos (*Component Palette*) clique no objeto



(*button* localizado na página *Standard*) e clique no formulário. Na janela da *Object Inspector* clique na página *Events* e clique duas vezes sobre a ação *OnClick* e insira o seguinte código:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Color := clGreen;
end;
```

Reparando no código completo da aplicação, veremos:

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;

type
  TForm1 = class(TForm)
    Button1: TButton;           Um novo dado foi aqui inserido
    procedure Button1Click(Sender: TObject);  Declarado um novo método
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);  O código do novo método
begin
  Form1.Color := clGreen;
end;

end.
```

O novo objeto *TForm1* agora apresenta um campo *Button1* - o botão que você adicionou ao formulário. *TButton* é o tipo do objeto, e *Button1* é o objeto botão propriamente dito. Com o tempo você colocará novos componentes ao formulário.

Rode o projeto, clicando no botão  (*Run*), dê um clique no botão e veja o que acontece. Pare a aplicação fechando a janela com **Alt+F4**.

U' Só por curiosidade, salve este arquivo, feche-o e abra o arquivo *UNIT1.DFM* (com a opção **File | Open File...**) notaremos que o *Delphi* criou um arquivo com todas as propriedades dos objetos criados e que a declaração do objeto **Form1** engloba todos os outros, noções de **Encapsulamento**.

Renomeando os objetos e os componentes

Você sempre deve utilizar a janela do *Object Inspector* para renomear os objetos criados. Por exemplo, o nome padrão do formulário é *Form1* mude a propriedade *Name* para **fCores**. O *Delphi* se encarregará de mudar qualquer referência que existia ao *Form1*. Então o código apresentará a seguinte modificação:

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;

type
  TfCores = class(TForm)                                Aqui foi modificado
  Button1: TButton;
  procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  fCores: TfCores;                                       Aqui foi modificado

implementation

{$R *.DFM}

procedure TfCores.Button1Click(Sender: TObject);        Aqui foi modificado
begin
  Form1.Color := clGreen;                                Aqui não !!!
end;

end.
```

U' O *Delphi* modificará apenas os códigos gerados automaticamente pôr ele. Os códigos para a ação *OnClick* foram gerados por você e o *Delphi* não os modificará. Cabe a você a manutenção neste caso. Isto foi idealizado para preservar o conteúdo original do seu código.

```
procedure TfCores.Button1Click(Sender: TObject);  
begin  
  fCores.Color := clGreen;  
end;
```

Capítulo II

Conhecendo o Delphi

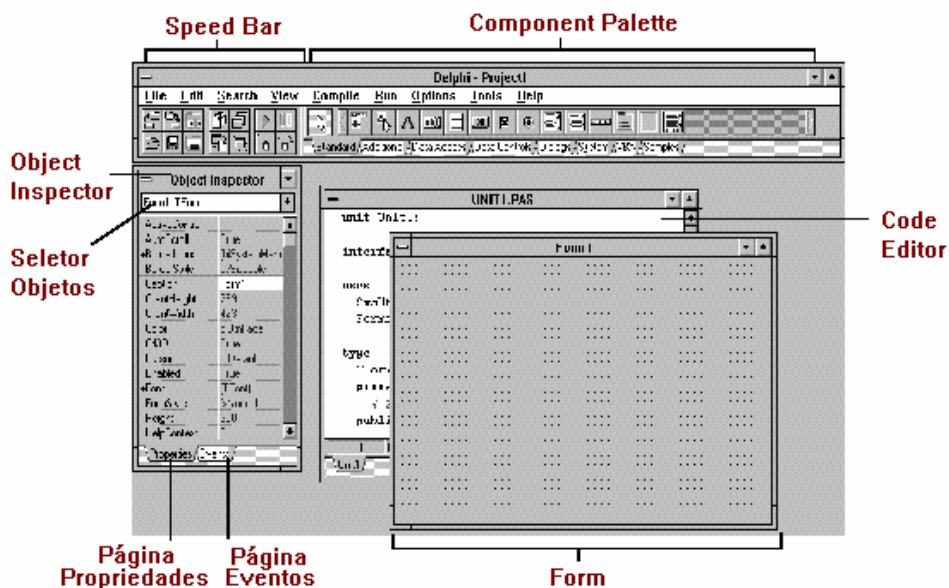
Se você teve algumas dúvidas no capítulo anterior sobre a área de trabalho do *Delphi* não se preocupe, neste capítulo você poderá saná-las completamente, também será mostrado o método de estrutura de aplicações Client/Server.

U' Caso você seja usuário do *Delphi 1.0* na barra de menu selecione a opção **H**elp e **I**nteractive Tutors, você receberá uma aula *On-Line* sobre a nova área de trabalho.

Os elementos da interface *Delphi* foram divididos do seguinte modo:

Elementos Visíveis

O ambiente de trabalho do *Delphi* é formado por objetos que estão visíveis tão logo que o aplicativo seja iniciado formando a área de trabalho.



Visão Geral dos objetos visíveis do Ambiente Delphi

Form

Os formulários (objeto *Form*) são os pontos centrais para o desenvolvimento *Delphi*. Você se utilizará deles para desenhar sua comunicação com o usuário, colocando e organizando outros objetos. Estes objetos são arrastados da *Component Palette*, mostrada na janela localizada acima.

Você pode imaginar que o formulário é um objeto que contém outros objetos. Sua aplicação ficará localizada em um formulário principal e este interagirá com outros formulários criados. É possível aumentar, mover ou ocupar completamente a tela do monitor, ou até mesmo ultrapassá-la. Um formulário básico inclui os seguintes componentes:

- ✓ Controles de menu;
- ✓ Botões de maximização e minimização;
- ✓ Barra de título; e
- ✓ Bordas redimensionáveis.

O código gerado, na área conhecida como *Code Editor*, fica exatamente atrás do objeto formulário, clique na barra de notas, em *Unit1*, se alguma coisa for desconhecida para você, leia maiores explicações no **Capítulo I**.

U' É possível enviar um formulário para a impressora, para isto existem duas maneiras:

1. Tipo um *PrintScreen* de Tela, coloque o seguinte comando "[Nome do formulário].Print;" no evento *onShow* do formulário; ou


2. Para imprimir um formulário no tamanho de um papel A4, através do uso de comandos da biblioteca *Printer*, veja o **Apêndice D** para maiores detalhes.

Code Editor

O editor de códigos providência total acesso ao código gerado pelo projeto, incluindo alguns dos mais poderosos recursos para a edição. Pode ser selecionado tipos de cores para os elementos do programa (como por exemplo comentários, palavras reservadas, códigos assembler, ...) para tanto a partir do menu principal entre em **Tools | Options...**, localize a página **Colors**.

U' Para outras informações adicionais sobre o modo de usar este editor, procure referências no *Help OnLine* no tópico **Code Editor**.

Ao ser aberto um novo projeto, o *Delphi* gera automaticamente na página do **Code Editor** uma *Unit* com o arquivo código (.PAS). Para ver o código de uma *Unit* em particular, simplesmente *Click* na tabulação de página. O **Code Editor** mostrará sempre o nome do arquivo corrente ativo na tabulação de página.

U' É possível alternar entre o objeto **Form** e a **Code Editor** através do pressionamento da tecla **F12**, do botão  (*Toggle Form/Unit*) da *SpeedBar*, ou ainda através das opções do menu **View | Toggle Form/Unit**. (curiosidade: o acesso rápido através da tecla Alt + Letra sublinhada para esta opção está marcado sobre a letra **G**)

Component Palette

Componentes (ou objetos) são os elementos que você usará para trabalhar com a aplicação. Foram incluídos objetos em várias páginas, tais como caixas de diálogos e botões, a palheta inclui também alguns espaços em branco para ser permitida a adição de novos objetos. Alguns objetos não serão visíveis enquanto a aplicação estiver executando, eles fazem parte do serviço da **DDE** (*Dynamic Data Exchange*).

Os objetos da palheta foram divididos em grupos de funcionalidade em diferentes páginas. Por exemplo, os objetos que representam as janelas tradicionais do *Windows*® (tais como fontes, palheta de cores, ...) foram colocados na página *Dialogs* da palheta.

Você poderá criar seus próprios objetos como também instalar outros já prontos, para isso foi colocado os espaços vazios. Por exemplo poderá ser instalado novos controles e objetos do **Visual Basic**® 4.0 (Objetos OCX - ou para os portadores do *Delphi 1.0* o do **Visual Basic**® 3.0 os objetos VBX). Uma das principais vantagens da **POO** é que muito em breve deverá ser colocado no mercado pacotes de objetos prontos para serem integrados aos sistemas o que facilitará ainda mais o desenvolvimento e a manutenção dos mesmos.

Object Inspector

Providência a conexão entre a interface visual e o código. É Composto por duas páginas *Properties* (propriedades) e *Events* (Eventos) que mostrará as propriedades e eventos do objeto selecionado.

Disponibiliza um fácil caminho para a personalização dos objetos. Você usará a página de **Propriedades** para personalizar os objetos colocados no formulário (inclusive o próprio formulário), e a página de **Eventos** para gerenciar a navegação entre certas partes do código do programa.

O seletor de objetos (*Object Selector* - localizado em um objeto do tipo *ComboBox* no topo do *Object Inspector*) mostra o nome e o tipo de todos os componentes do formulário corrente (inclusive o próprio). Você pode usar o seletor de objetos para localizar facilmente qualquer objeto no formulário.

SpeedBar

Contém os botões mais freqüentemente utilizados. Fornecendo deste modo um atalho de navegação ao menu principal do *Delphi*.

É possível a personalização da *SpeedBar* colocando nela os caminhos do menu principal que você mais utiliza, bastando para isso:

1. Redimensione a *SpeedBar*. Para tanto posicione o cursor do mouse sobre o ponto de encontro da *SpeedBar* com a *Component Palette* conforme o desenho abaixo:




2. Quando o cursor do mouse mudar de formato, clique o botão esquerdo do mouse e arraste abrindo a área da *SpeedBar*.
3. Clique com o botão direito do mouse na área aberta, apareça um menu *PullDown* contendo entre outras opções a opção *Properties*, selecione-a.
4. As categorias e os comandos são divididos de acordo com o menu, clique em cima dos comandos disponíveis e arraste-os para a área aberta de acordo com a sua necessidade, para retirar os botões da *SpeedBar* faça o processo inverso.

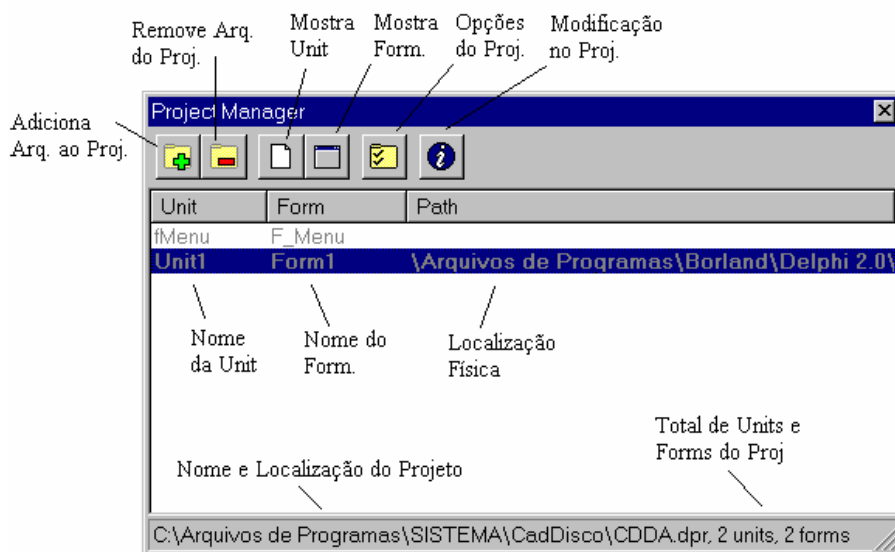
Elementos não Visíveis

Alguns elementos não estão prontamente visíveis quando o *Delphi* é iniciado mas você poderá ter acesso a eles bastando para isso selecionar a opção na barra de menu.

Project Manager

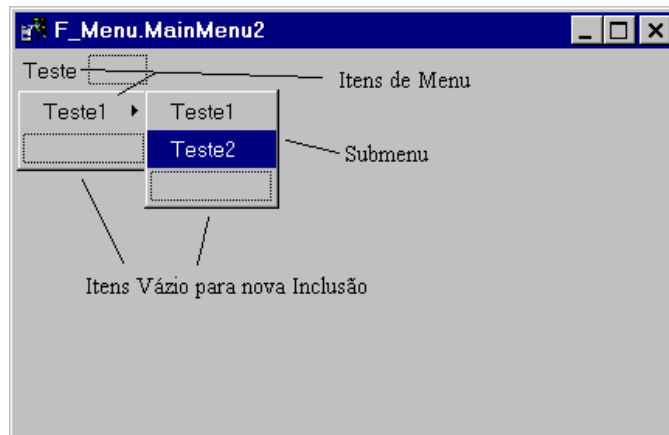
O Gerenciador de Projetos contém uma lista de formulários ou unidades utilizados pela aplicação, e serve para controlar estes formulários ou unidades, adicionando-os ou removendo-os do projeto, organizando as opções do projeto, entre outros.

U' Você também poderá colocar um botão para iniciar o Gerenciador de Projetos através da *SpeedBar* .



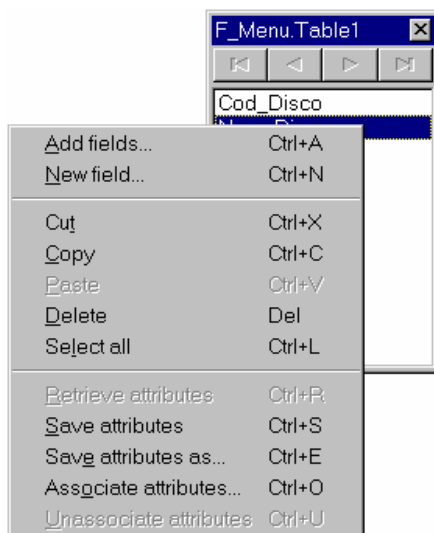
Menu Designer

O *Menu Designer* permite a criação de menus para os seus formulários. Você pode criar menus ou menus do tipo *pulldown* através dos objetos *MainMenu* ou *PopupMenu* (localizado na página *Standard* na *Component palette*). A criação completa de **Menus** será colocada de forma mais abrangente no **Capítulo IV**.



Fields Editor

Para o *Delphi* é possível editar e modificar as propriedades de quaisquer campos dos objetos de tabelas associadas ao banco de dados, a *Fields Editor* em conjunto com a *Object Inspector* controlam o modo de mostrar determinados campos de arquivos, é importante lembrar que esta modificação não afetará os campos da tabela, apenas para o formulário ativo em questão. Sua utilização efetiva será mostrada a partir do **Capítulo V**.



Add fields... é responsável pela adição de definições de campos da tabela, é possível inserir um ou mais campos, dependendo de sua utilização para o formulário.

New field... permite a criação de um novo campo, este pode ser a derivação de um ou mais campos da tabela.

Cut envia para área de transferência e elimina todas as definições do campo selecionado.

Copy copia para a área de transferência todas as definições do campo selecionado.

Paste recebe da área de transferência todas as definições do campo selecionado criando-o.

Delete exclui quaisquer definição para os campos.

Select all seleciona todas as definições dos campos.

Retrieve Attributes atualiza os atributos do campo selecionado com os campos do dicionário de dados.

Save attributes salva os atributos do campo selecionado para o dicionário de dados.

Save attributes as... salva os atributos do campo selecionado para o dicionário de dados permitindo a renomeação do campo.

Associate attributes... faz a associação dos atributos do campo selecionado com determinado campo do dicionário de dados.

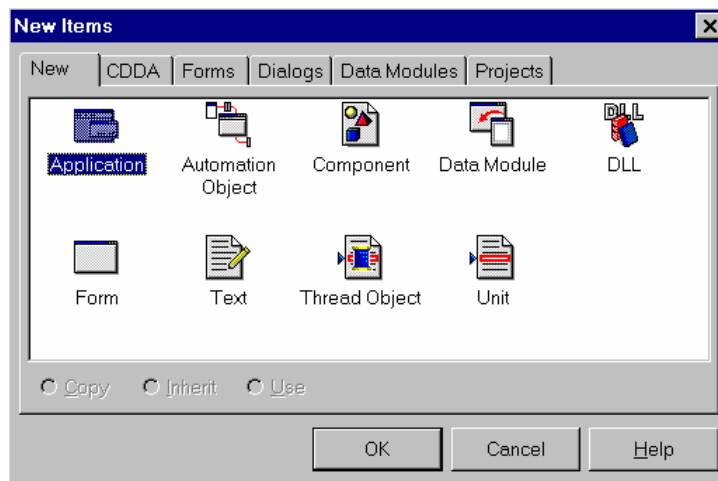
Unassociate attributes remove a associação dos atributos do campo selecionado com determinado campo do dicionário de dados.

Repositório de Objetos

O **Repositório de Objetos** do *Delphi 2.0* armazena e gerencia os objetos da aplicação: Formulários, *Data Modules*, geradores *experts*, e DLL (*Dinamic Linked Library* - Bibliotecas de acesso dinâmico). Na essência, ele centraliza as localizações dos objetos envolvidos agrupando-os. A proliferação dos objetos no repositório incrementa as seguintes vantagens:

- Suporte a equipe de desenvolvimento para referência aos objetos da rede.
- Uma customização de todo o desenvolvimento em grupos lógicos de objetos, facilitando o re-uso dos mesmos.

O *Delphi* possui diversas características quanto a sua utilização. Tem os *Tutors* e *Experts* que são as ferramentas responsáveis para guiar-nos através de técnicas, tais como, manipulação de componentes e criação de simples aplicações. Além disso o *Delphi* oferece uma coleção de modelos para formulários, janelas de diálogo e até mesmo aplicações completas na ferramenta *New Items*. A janela do *New Items* é sempre chamada automaticamente quando a opção **File** | **New...** do menu principal é executada.



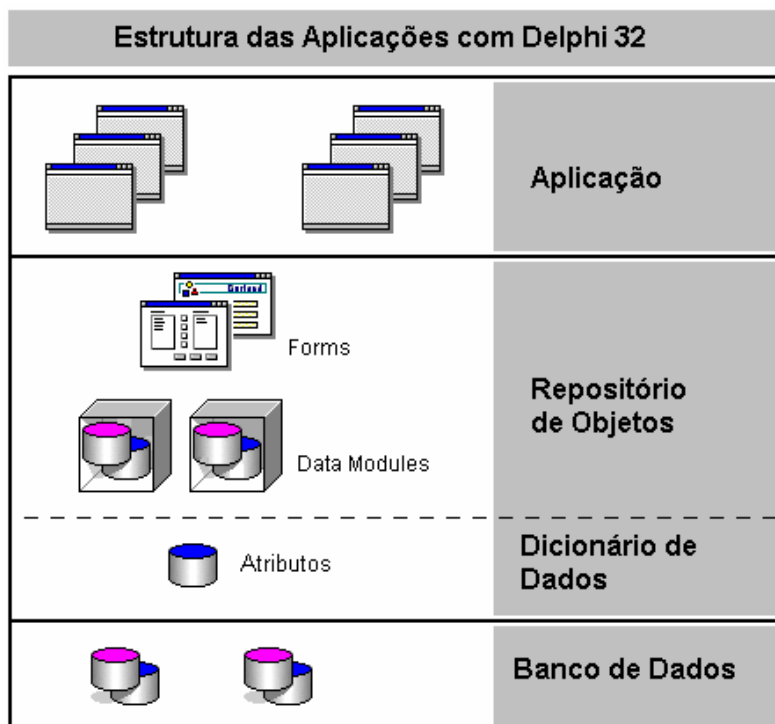
U'É possível para você criar novas janelas, ou projetos, automáticos no *Delphi*. Para a nossa sorte a **Borland** não esconde o jogo e mostra como se cria uma *template* visite o diretório **\\BORLAND\\DELPHI 2.0\\OBJREPOS**, o equivalente no *Delphi 1.0* é encontrado no diretório **\\DELPHI\\GALLERY**, todos os exemplos são auto-explicativos.

U'Para definir o projeto padrão que o *New Items* executará no início de cada projeto, clique com o botão direito acima da janela e escolha a opção **Properties** aparecerá as listas *Pages* e *Objects*, defina quaisquer dos objetos como *New Form* ou *Main Form*.

U'Observe que a segunda folha da janela *New Items* (terá o nome do seu arquivo com a extensão **.DPR**) é o seu projeto corrente, ou seja, uma nova janela poderá ser derivada, por característica de herança, de uma outra janela já existente.

Estrutura de Aplicações com o *Delphi 2.0*

Um dos objetivos de desenhar aplicações do tipo *Client / Server* para o desenvolvimento é a reutilização dos objetos, das regras de negócio e das telas do projeto. O *Delphi Client / Server Suite 2.0* é o único que implementa uma arquitetura incorporando a tecnologia **RAD** (*Rapid Application Development* - Desenvolvimento Rápido de Aplicações) com o desenvolvimento totalmente **OO** (*Object Orientation* - Orientado a Objetos) para a redução do tempo de desenvolvimento e manutenções improváveis. Adicionando, uma arquitetura aonde é possível a separação da **GUI** (*Grafic Unit Interface* - Unidade de Interface Gráfica), das regras de negócio lógicas e do desenho do banco de dados de acordo com o modelo representado abaixo:



Benefícios:

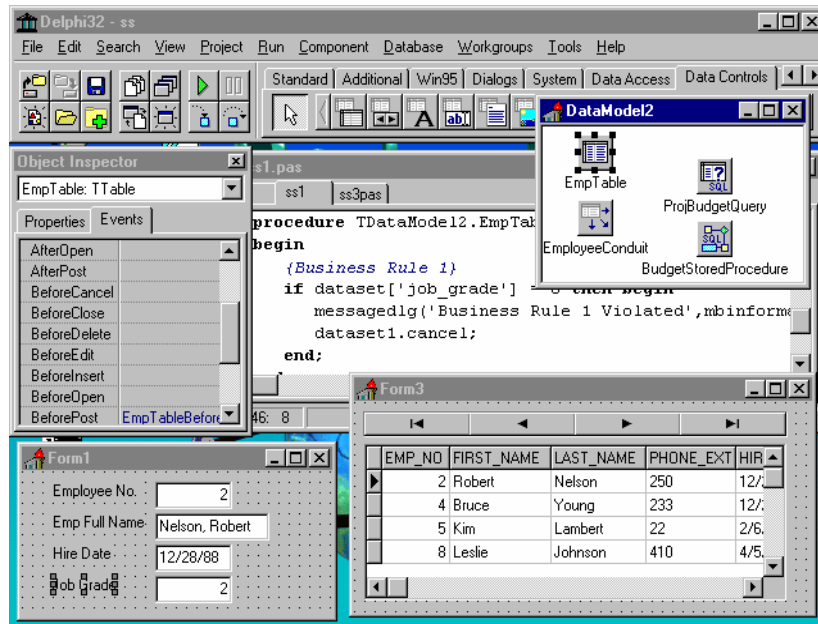
1. A separação do desenho da **GUI** (através dos objetos **Forms**) com a área de ligação lógica de dados (através dos objetos **Data Modules**) permite que se cause um menor impacto sobre ambas as áreas. As mudanças podem ser executadas nas telas de entrada ou nas ligações, independentemente, de acordo com os requerimentos do usuário.
2. A separação do desenho da **GUI** com a área de ligação lógica de dados realizada com uma certa habilidade aos eventos envolvidos, poderá não necessariamente deverá ser controlada por um habilidoso **DBA** (*Database Administrator* - Administrador de Banco de Dados), com suas fantásticas regras de negócio, podendo inclusive ser mantida por um “analista de informações”.

3. O desenho do banco de dados, a construção das metodologias de negócio e o desenho e a criação das janelas de entrada são efetivados dentro da aplicação. O desenvolvimento, então paralelamente, resultará em uma maior rapidez.
4. Herdando as janelas de entrada, em níveis de utilização, reduz-se drasticamente o processo de codificação, e em consequência, o processo futuro de manutenção. As mudanças lógicas das regras de negócio ou a incorporação de novos padrões, são feitas automaticamente para todos os objetos herdados.
5. O repositório de objetos, os formulários reusáveis e a utilização dos objetos **Data Modules**, envolve em eliminação da duplicação de códigos e de trabalhos com desenhos de janelas, e em consequência na redução da equipe de desenvolvimento.

Implementação efetiva

Objetos Data Module

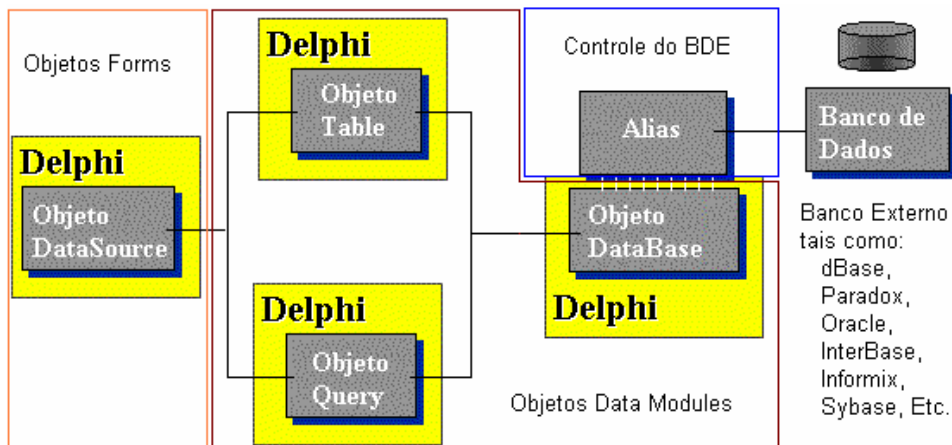
A partir do *Delphi 2.0* foi incorporado o uso de objetos conhecidos por **Data Module**, que servem para que suas aplicações providenciem um desenho centralizado da definição de acesso aos dados e das regras de negócio. Os objetos **Data Modules**, também podem ser separados por negócios lógicos (como exemplo por áreas: compras, vendas, estoque, etc.) formando caminhos de conexões simples.



- Os objetos **Data Modules** podem ser aplicados a objetos tais como **Tables**, **Stored Procedures**, ou **Queries** permitindo a centralização dos eventos envolvidos em antes e

depois da gravação, exclusão, inserção ou edição dos dados. E até mesmo na colocação de novos objetos de controle para maior facilidade.

- As relações de dados *Master / Detail* são definidas em menor quantidade. Possibilita então ao desenvolvedor criar aplicações do tipo *Client / Server* de forma mais fácil, rápida, e segura se utilizando das propriedades dos objetos **Datasources** ou utilizando o **Database Form Expert**.
- Os formulários das aplicações, podem ser ligados diretamente a um ou mais objetos **Data Module** para a propagação das regras de negócio sem a necessidade de execução de um código extra.
- Os objetos **Data Modules** são classes de objetos que pertencem a interação dos dados do *database server*. Isolando totalmente o acesso ao banco de dados com a aplicação *Client*, simplificando deste modo toda a manutenção realizada.
- O Acesso ao *Delphi* aos bancos de dados se processa da seguinte maneira:



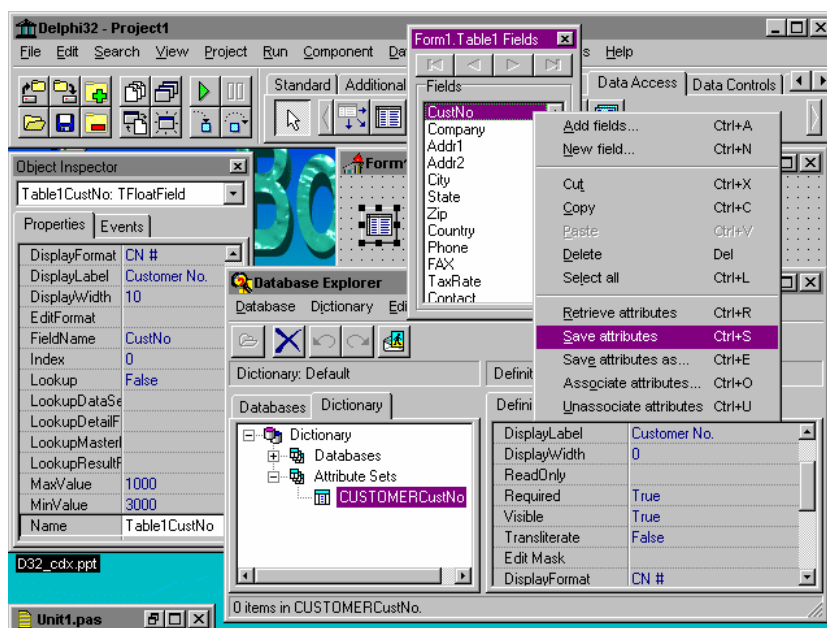
Distribuição Lógica da Aplicação:

Os objetos **Data Module** foram desenvolvidos para criar uma centralização lógica de todas as regras de negócio, separando a área de visão do usuário com a área do desenho do Banco de Dados. Este sólido fundamento de suporte são distribuídos em um n-número de aplicações e arquiteturas servidoras disponíveis, tais como **CICS** da **IBM**, **TopEnd** da **ATT**, **Tuxedo** da **Novell**, **Object Broker** da **Digital**, **IONA** da **Orbix** e muitas outras.

Dicionário de Dados Escalável

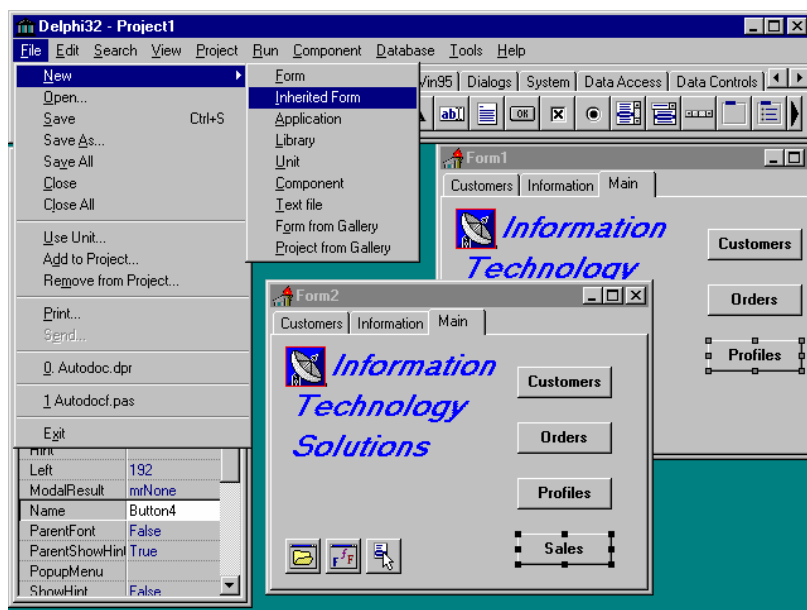
O **dicionário de dados** é utilizado para armazenar informações sobre o uso dos dados contidos nas suas tabelas. O dicionário deve ser como uma árvore genealógica trabalhando como um inspetor de modificações que permitem um armazenamento facilitado. O dicionário deve conter informações dos atributos dos campos tais como: valores mínimo, máximo e comuns (valores *default*), máscaras utilizadas, etc. A utilização do dicionário de dados traz as seguintes vantagens:

1. **Consistência:** Campos idênticos são armazenados centralmente no dicionário isto reduz o tempo de definição das duplicidades. Um desenvolvedor poderá criar os campos complementares com domínios e aplicações apropriadas.
2. **Redução do Tráfego da Rede:** O *Delphi* permite que a validação dos dados seja feita nas máquinas *client* ou no servidor. O Dicionário de dados permite que a manutenção dos atributos dos campos do lado *client* seja validado de forma eficiente reduzindo a necessidade do tráfego da rede.



Herdando os Formulários

O desenvolvimento de aplicações corporativas de uma forma padronizada é um fato de suma importância para as empresas envolvidas. Mas conseguir e manter este padrão é uma tarefa considerada praticamente impossível, já que as aplicações devem se modernizar na velocidade que o mercado de informática exige. Os formulários herdados do *Delphi* são simples extensões da programação orientada a objetos, conseguindo manter, de forma automática, os padrões e as modificações realizadas nos projetos. E em conjunto com o **Repositório de Objetos**, padroniza, organiza e centraliza os formulários resultando em modificações de curtíssimo tempo.



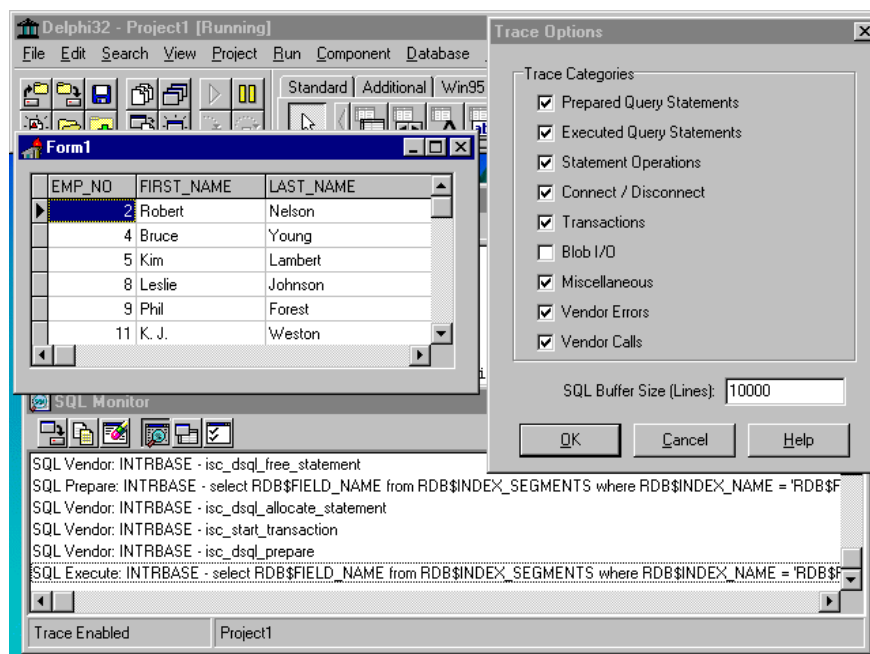
Ferramentas Auxiliares de SQL

Para o trabalho com bases de dados padrão **SQL** (*Structure Query Language* - Linguagem estruturada de Consultas), o *Delphi* conta com as seguintes ferramentas **RAD** que auxiliam ao desenvolvimento.

Monitor SQL

Um **monitor SQL** é uma ferramenta para testes, depuração e execução de consultas **SQL** em aplicações *Client / Server*. Isto resulta em um aumento da produtividade de desenvolvimento e melhor performance da aplicação.

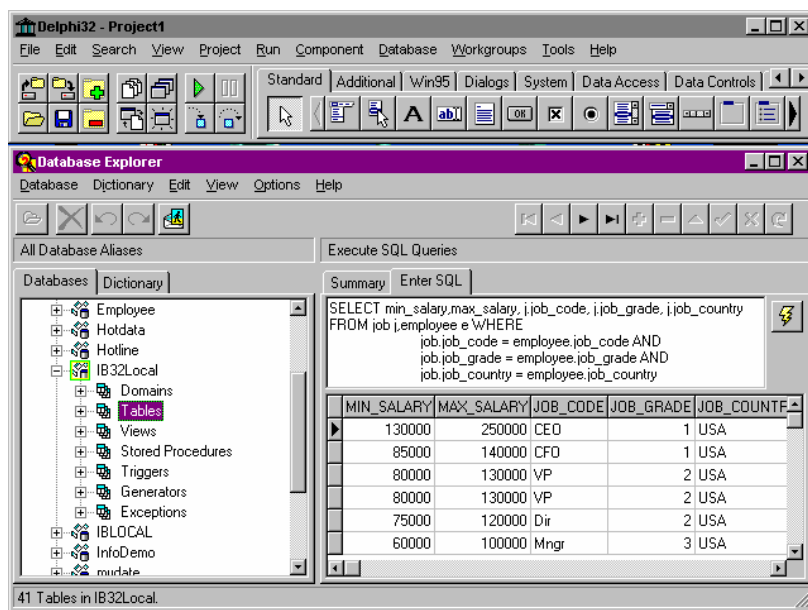
O **monitor SQL**, intercepta as chamadas entre as máquinas *client* e o servidor. Esta informação auxilia ao desenvolvedor em problemas relacionados às declarações **SQL** e otimiza este tipo de transação. Uma série de caminhos de interceptações podem ser traçados, dependendo da necessidade do desenvolvedor, para que as informações a serem colocadas em um relatório *on-line* sejam as mais imprescindíveis o possível. É possível inclusive salvar e imprimir o relatório gerado para consultas ou testes posteriores.



SQL Explorer

A ferramenta **SQL Explorer** providencia uma informação centralizada do gerenciamento das demandas da base de dados; tais como, suporte a modificação e criação de tabelas, sinônimos, procedimentos de gravação, *triggers* (gatilhos disparados pelo banco) e execução das regras de negócio interativas do **SQL**. Uma ferramenta gráfica que proporciona um esquema de integridade da base da dados e contém as ferramentas essenciais para os administradores de bancos de dados.

O **SQL Explorer**, unicamente para o *Delphi*, administra de forma intuitiva e fácil o banco de dados. A simplicidade de uso da interface gráfica é um perfeito caminho para representar o complexo relacionamento que existe no banco de dados do servidor. Apresenta um esquema para informações em bancos como **Oracle**[®], **Sybase**[®], **InterBase**, **Informix**[®], **DB2**[®] e outros. O desenvolvedor poderá trilhar campos, tabelas e procedimentos do banco dentro da construção da aplicação *Delphi* rapidamente, podendo ser direcionado para múltiplos servidores e múltiplos bancos.



O **SQL Explorer**, também administra o **Dicionário de Dados**. Sua interface de uso simplificado permite facilmente definir novos domínios para os atributos dos campos e associação entre tabelas.

InterBase NT - Banco de Dados Relacional

O *Delphi Client / Server Suite 2.0* inclui uma licença para dois usuários do uso do banco de dados **InterBase NT**. Desenvolvedores podem criar em máquinas *standalone* aplicações usando este poderoso banco de dados (concorrente de bancos como **Oracle**®, **Sybase**®, **Informix**®, **DB2**® e outros). Com o crescimento do volume de dados e do tamanho da aplicação, ambos, o **InterBase** e o **Delphi** conseguem interagir de forma harmoniosa.

O **InterBase** é um banco de dados de alta performance produzido pela **Borland**, como plataforma para **SQL Server**. Está disponível em mais de 15 sistemas operacionais incluindo: **DOS e Windows**® 3.1, **Windows**® 95, **Windows**® NT, **NetWare**®, **SCO**®, **Sun OS**®, **Sun Solaris**®, **HP-UX**®, **IBM AIX**®, **SGI IRIX**®, etc.

O **InterBase** é um banco a nível **ANSI SQL 92**, suportando eventos programados e exceções ocorridas no modelo por acesso de múltiplos usuários. Oferece chaves de controle a nível de registros para arquiteturas Multi-Gerenciais causando um performance muito superior a uma leitura das operações de banco, em contrário das leituras de bloqueio de operações escritas realizadas por outros bancos.

A versão local do **InterBase**, disponível apenas com a cópia *Client / Server Suite 2.0*, providencia aos desenvolvedores um caminho rápido para o desenvolvimento de protótipos e de sistemas com um banco de padrão **ANSI 92 SQL**. Esta versão propicia as mesmas funcionalidades da versão multi-usuário para **NT** e **Unix**, incluindo controles de transações, procedimentos de gravação (*stored procedures*), uso de *triggers* (gatilhos disparados do banco), ou eventos de

alerta. Imagine o desenvolvimento de um grande sistema sendo realizado em um *Laptop* dentro de um trem, avião ou até mesmo na frente do cliente, apenas o acesso ao banco de dados final é que será modificado.

Utilizando o *Delphi Client/Server Suite 2.0*, desenvolvedores poderão conceber e desenhar protótipos e testar a aplicação final em uma única máquina. O **InterBase** oferece um excepcional acesso a interface gráfica do **Windows**[®], incluindo a configuração das propriedades, um perfeito gerenciador de bancos nativo 32 bits, total interatividade com ferramentas SQL, e uma completa documentação em formato de *Help* do **Windows**[®] (arquivos .HLP).

Capítulo III

Projeto Piloto

É óbvio que fica mais simples o aprendizado de uma nova ferramenta quando se faz algum tipo de aplicativo, principalmente um que seja útil, então ao longo deste estudo, iremos desenvolver um aplicativo destinado ao Cadastro de Compact Disc (CD's). Todas as pessoas hoje em dia tem montes de CD's, virou uma espécie de febre, então, porque não fazer um sistema para cadastrá-los e controlá-los, quanto tempo você já perdeu pensando em qual deles está aquela música que você quer ouvir? Ou uma capa que seu filho rasgou, que tal imprimi-la novamente? E no capítulo multimídia aprenderemos um método para tocar o CD.

Para darmos partida ao nosso primeiro aplicativo (Projeto Piloto), definiremos inicialmente as nossas necessidades:

1. Permitir o cadastro completo e a consulta aos CD's;
2. Ser possível separar os CD's em categorias, facilitando deste modo a busca e o armazenamento;
3. Quanto as músicas deve ser permitido o cadastro do autor e o tempo de duração;
4. Permitir a inclusão da foto da capa do CD; e
5. Comportar relatórios de conferência e reimpressão da capa.

O acesso *Delphi* a arquivos pode ser feito através de duas maneiras local ou remoto, sendo a segunda apenas possível pela cópia *CLIENT-SERVER* é voltadas a bases de dados mais complexas como *ORACLE*[®] ou *SYBASE*[®], inicialmente, restringiremos o nosso estudo a base de dados locais conseguida através da versão *Desktop*. Lembre-se que à versão *Desktop* contém acesso a criação e a definição de bases *dBase* e *Paradox*, além de outras conseguidas através de *ODBC*.

Com base no que foi sugerido acima, vamos definir as tabelas:

CATEGORIA

Objetivo: Dados das categorias do CD.

Campos: SIGLA DA CATEGORIA - Abreviação da descrição da categoria.

DESCRIÇÃO DA CATEGORIA - Descrição da Categoria.

BÁSICO

Objetivo: Dados iniciais do CD.

Campos: CÓDIGO DO DISCO - Código do CD, encontrado na própria capa.

NOME DO DISCO - Nome do CD.

TIPO DO DISCO - Tipo de Gravação do CD: AAA, AAD, ADD ou DDD.

FOTO DA CAPA - Armazenará a foto da capa do CD.

SIGLA DA CATEGORIA - Ligação para o código da Categoria.

MÚSICAS

Objetivo: Dados das músicas do CD.

Campos:

CÓDIGO DO DISCO - Ligação com o CD.

NUMERO DA FAIXA - Número da faixa.

NOME DA MÚSICA - Título da música.

NOME DO AUTOR - Nome do autor da música.

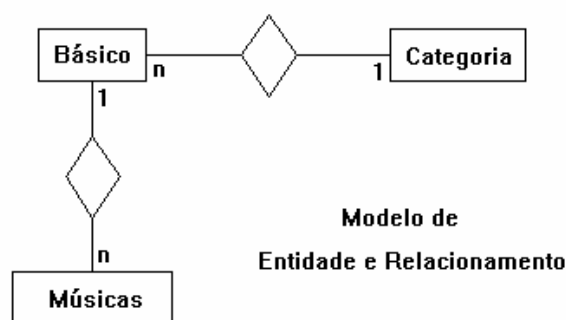
TEMPO DA MÚSICA - Tempo de duração da música MMSS.

Criando o Modelo Relacional

Para criar um modelo relacional simples e trabalhar com bases locais você pode optar por dois modos do tipo *dBase* ou *Paradox*, particularmente eu prefiro trabalhar com *Paradox*, mas você rapidamente notará que tanto faz, trabalhe com aquela que você se sintá mais a vontade.

U' Escolha a base de acordo com o porte do sistema: Para sistemas pequenos e simples e com poucos dados use o *dBase*, para sistemas médio, multi-usuário, com uma boa quantidade de dados escolha a base *Paradox* e para sistemas complexos em várias plataformas e acessos multi-usuário opte por *InterBase*.

Vamos agora visualizar um modelo que demonstrará como as tabelas deverão se relacionar no sistema, este modelo também facilitará as relações e a integração dos formulários do sistema quando construirmos nossas consultas e relatórios.



No modelo de entidade e relacionamento acima lê-se:

- 1 registro da entidade **Básico** se relaciona com 1 registro da entidade **Categoria** enquanto que 1 registro da entidade **Categoria** se relaciona com n registros da entidade **Básico**,
- 1 registro da entidade **Músicas** se relaciona com 1 registro da entidade **Básico** enquanto que 1 registro da entidade **Básico** se relaciona com n registros da entidade **Música**.

Com o MER nas mãos já se tem a idéia de como deve ficar as estruturas das tabelas, então, vamos queimar um pouco de neurônios.

Trabalhando com DataBase Engine Configuration

Criando o Alias

O **ALIAS** é simplesmente um apelido (sinônimo) a ser dado para o banco de dados, este apelido permitirá que no lugar de falarmos para ao *Delphi* que a nossa base se encontra em C:\SISTEMA\... ou D:\DESENV\SISTEMA\..., simplesmente digamos se encontra em AliasX ou AliasY, isto facilitará o seu trabalho quando você por exemplo quiser modificar o diretório do sistema, basta mudar o endereço do **ALIAS** e não sair modificando vários formulários.

U' Outra vantagem em se criar um Alias está na mudança da base, basta reapontarmos o Alias para outra base que o sistema automaticamente verá estas novas informações. Lembre-se que para isto ser possível é necessário que o nome das tabelas e dos campos sejam necessariamente os mesmos, incluindo o tamanho e o tipo (no apêndice B é encontrado uma tabela para a conversão das diversas bases de dados).

Para trabalhar com **ALIAS** o caminho mais interessante é com o *Database Engine*



Configuration, no arquivo de programas do *Delphi* dê um duplo clique sobre o ícone *BDE Configuration*, a configuração do banco de dados se divide em várias páginas:

- ✓ **Drivers** - Controla os arquivos de acesso locais e ODBC dos bancos de dados utilizados;
- ✓ **Aliases** - Controle dos sinônimos dos sistemas;
- ✓ **System** - Define os recursos do *Windows* que serão alocados pela aplicação;
- ✓ **Date** - Especifica os formatos utilizados para campos tipo data;
- ✓ **Time** - Especifica os formatos utilizados para campos tipo hora; e
- ✓ **Number** - Especifica os formatos utilizados para campos tipo numérico.

Se atualmente você estiver utilizando a versão de desenvolvimento, os *drivers* que aparecerão serão: dBase, InterBase e Paradox, na versão *Client/Server* além desses serão colocados: Oracle, Informix, SyBase entre outros.

Mude para a página *Aliases* e click no botão **New Alias**, informe:

New alias name: AliasDisco

Alias type: STANDARD

Botão **OK**

Path: C:\SISTEMA\CADDISCO

Default Driver: Paradox

Neste momento o seu **ALIAS** AliasDisco foi criado para a banco de dados Paradox, no formato Padrão localizado no diretório C:\SISTEMA\CADDISCO, no menu principal escolha a opção **F**ile e **S**ave, o *BDE* salvou o seu arquivo de configuração chamado IDAPI.CFG.

U' Crie com o Gerenciador de Arquivos o diretório C:\SISTEMA\CADDISCO, aonde será localizado o sistema.

U' Para a base dBase a única diferença seria o comando **Default Driver:** DBASE.

Encerre o *Database Engine Configuration*.

Trabalhando com DataBase DeskTop



Database Desktop
Version 7.0
Copyright © 1992-1996
Borland International, Inc.
All Rights Reserved.

Criando o Banco de Dados via Estrutura

Para criar suas tabelas, dispõe-se de duas maneiras: o modo declarações em *SQL* ou pela janela de estrutura, inicialmente, utilizaremos a janela de estrutura, pois além de mais simples é mais prático, a menos que você trabalhe com bases de acesso remoto, evite o modo declarações em *SQL* para manusear a estrutura de tabelas, principalmente tabelas simples como é o caso do *Paradox* ou o *dBase*, apesar que existem muitos fanáticos por *CREATE TABLE*, *ALTER TABLE* e *DROP*'s. Em seguida mostrarei como criar as mesmas tabelas utilizando o método *SQL*.

U' Lembre-se de apagar as tabelas antes de criá-las novamente. Senão não será possível executar uma única declaração *SQL*.

Retorne ao *Delphi*, no menu principal escolha a opção **T**OOLS|**D**ataBase **D**esk**T**op, maximize a tela para permitir uma melhor visualização.

Com o nosso sinônimo (**ALIAS**) criado iremos agora definir as nossas tabelas. Inicialmente vamos definir como nossa área de trabalho: No menu principal, opção **F**ile|**W**orking **D**irectory..., na opção **A**liases: selecione **AliasDisco**, note que a opção **W**orking **D**irectory será automaticamente modificada para **:AliasDisco**., finalmente confirme clicando no botão OK. O diretório apontado pelo Alias, agora será o **default**, ou seja, tudo o que fizermos será apontado para o diretório.




No menu principal, opção **F**ile, opção **N**ew, e opção **T**able..., será mostrada uma janela com os tipos de possíveis repositórios de tabelas no **Table Type** escolha a opção **Paradox 7** e clique no botão **OK**. Insira os seguintes campos:

Field Name	Type	Size	Key
COD_DISCO	N		*
NOM_DISCO	A	60	
TIP_DISCO	A	3	
FOT_CAPA	B	3	
SIG_CATEG	A	2	

U' Dos campos, o único que merece uma explicação é FOT_CAPA ele foi escolhido neste formato (Binary) pois guardará uma imagem BitMap (extensão **.BMP**) da Capa do CD, todos os outros campos são caracteres alfanuméricos, com a exceção do COD_DISCO que é um campo Numérico e chave.

Para esta tabela precisamos ainda criar um índice secundário, para tanto na opção **Table properties:** chame a opção **Secondary Indexes** e clique no botão **Define...**, marque no campo

Nom_Disco (na lista *Fields*) e clique no botão  o campo passou para a lista *Indexed Fields*, clique no botão **OK** e digite SI_NomDisco para o nome do índice e clique no botão **OK**.

Para salvar sua tabela clique no botão **S**ave as... na opção **N**ome do Arquivo: insira o nome da tabela - Basico.

Crie agora as seguintes tabelas:

1. Categor


Field Name	Type	Size	Key
SIG_CATEG	A	2	*
DES_CATEG	A	40	

2. Musica

Field Name	Type	Size	Key
COD_DISCO	N		*
NUM_FAIXA	N		*
NOM_MUSICA	A	60	

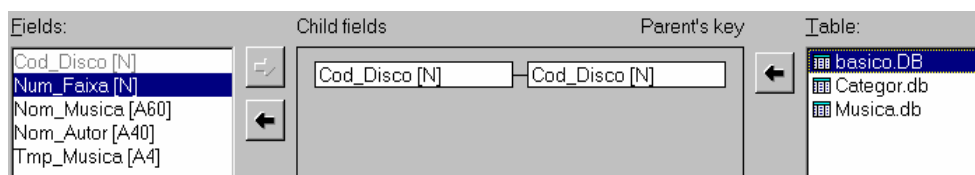
NOM_AUTOR	A	40
TMP_MUSICA	A	4

Criando os Relacionamentos via Estrutura

Os relacionamentos entre as tabelas poderia ter sido criado no momento da construção das mesmas, mas acredito que deste modo seja mais simples, a partir do menu principal, opção **File**, opção **Open**, e opção **Table...**, será aberto uma janela com todas as tabelas, clique na tabela **BASICO** e sua estrutura será mostrada, clique no botão  **Restructure**, ou a partir do menu principal, opção **Table**, opção **Restructure...**, na opção **Table properties**: alterne para a opção **Referential Integrity** e clique no botão **Define...**, no lado esquerdo (Lista **Fields**) dê um duplo clique no campo **SIG_CATEG** e no lado direito (Lista **Tables**) dê um duplo clique na tabela **CATEGOR.DB**, clique no botão de OK para confirmar e será solicitado o nome para o índice.

Para o nome do índice crie um padrão de FK_ + {nome do campo} + {nome tabela pai}, uma padronização dos nomes dos índices facilita a procura futuramente, então para o nosso índice crie FK_SigCateg_Basico.

Crie agora o outro relacionamento entre a tabela **MUSICA** e **BASICO** conforme a figura abaixo:




U'Também é possível acessar o *Database Desktop* através do ícone  localizado no grupo de trabalho **DELPHI**.

Encerre o *Database Desktop* e retorne ao *Delphi*, ou apague as tabelas e...

Criando o Banco de Dados via SQL


Para os fanáticos por declarações *SQL*, vamos criar a mesma base de dados via *SQL*, se você ainda não o fez, observe no tópico anterior como colocar o **ALIAS** na área de trabalho. Isto será de muita utilidade quando formos salvar o nosso trabalho.

No menu principal, opção **File**, opção **New**, e opção **SQL File**, será mostrada uma janela para ser digitada a declaração *SQL*, observe na barra de comandos o botão  **Select Alias**, ou a partir do menu principal opção **SQL/Select Alias...**, apenas observe que a área de trabalho **Work** já está selecionada, clique em OK ou Cancel sem fazer nenhuma modificação, observe que o título da janela é **SQL Editor :WORK:<Untitled>**.

Digite o seguinte na janela:

Create Table Basico

```
(  
  COD_DISCO Numeric(4,0),  
  NOM_DISCO VarChar(60),  
  TIP_DISCO Char(3),  
  FOT_CAPA Blob(3,2),  
  SIG_CATEG Char(2),  
  Primary Key(COD_DISCO)  
);
```

Clique no botão  **Run SQL**, ou no menu **SQL/Run SQL**, ou ainda pressione **F8**, após executado o comando a tabela será criada. Salve o SQL com a opção **File/Save** (digite BASICO .SQL)

Abra novas janelas e crie o resto das tabelas:

Create Index SI_NomDisco on Basico (NOM_DISCO);

Criando o índice secundário da BASICO

Create Table CATEGOR

Criando a Tabela CATEGOR

```
(  
  SIG_CATEG Char(2),  
  DES_CATEG VarChar(40),  
  Primary Key(SIG_CATEG)  
);
```

Create Table MUSICA

Criando a tabela MUSICA

```
(  
  COD_DISCO Numeric(4,0),  
  NUM_FAIXA Numeric(2,0),  
  NOM_MUSICA VarChar(60),  
  NOM_AUTOR VarChar(40),  
  TMP_MUSICA Char(4),  
  Primary Key(COD_DISCO, NUM_FAIXA)  
);
```

O SQL para o Paradox não consegue executar a criação de índices referenciais (ou “Constraints”) então para criar este tipo de índice crie-o através da janela de estrutura conforme mostrado anteriormente.

Observações da utilização do SQL com o dBase

Infelizmente o *dBase* não suporta a cláusula *Primary Key* que permite a criação dos índices muito menos a criação de chaves estrangeiras. Quanto as chaves estrangeiras não se preocupe pois o *Delphi* consegue manipular relacionamentos entre as tabelas mesmo que elas não estejam vinculadas.

Caso você esteja utilizando este tipo de base os passos para as criações das tabelas são os mesmos mas corte as cláusulas *Primary Key* é necessário criar os índices separadamente. Abra uma nova declaração SQL, idêntica a anterior, e digite o seguinte na janela:

Create Index PK_Unica on Basico (COD_DISCO);

U' O comando correto para este índice deveria ser “**Create Unique Index PK_Unica on Basico (COD_DISCO);**” mas isto provoca um erro colocando que não é possível esta declaração, então coloque a cláusula **UNIQUE** através da janela de estrutura.

U' Se você esqueceu de ativar o **Select Alias** coloque “**Create Index Cod_Disco on “Basico.dbf” (COD_DISCO);**”

As outras declarações são:

Create Index SI_NomDisco on Basico (NOM_DISCO);	Criando o índice secundário
Create Index FK_SigCateg_Basico on Basico (SIG_CATEG);	Criando a chave estrangeira
Create Index PK_Unica on Categor (SIG_CATEG);	Criando a chave primária
Create Index FK_CodDisco_Musica on Musica (COD_DISCO);	Criando a chave estrangeira

U' O comando correto para a criação da chave dupla da tabela MUSICA seria “**Create Index Chv_Unica on Musica (COD_DISCO, NUM_FAIXA);**” mas novamente é provocado um erro mostrando a impossibilidade de execução do comando, então crie este índice através da janela de estrutura.

U' É facilmente percebido que o *DataBase DeskTop* até que tenta colocar todas as bases de dados compatíveis com a linguagem SQL, mas infelizmente ainda não foi nesta versão.

U' Para as tabelas do tipo ORACLE, INTERBASE, SYBASE e MS SQL Server a criação das tabelas podem ser feitas tanto pelo modo de estrutura quanto pelas declarações SQL mas lembre-se que a alteração das mesmas só poderá realizar-se através do modo de declarações SQL. Aqui vão alguns exemplos destas declarações:

Create Table MUSICA (COD_DISCO Numeric(4,0), NUM_FAIXA Numeric(2,0), NOM_MUSICA VarChar(60), TMP_MUSICA Char(4), Constraint PK_Unica Primary Key(COD_DISCO, NUM_FAIXA), Constraint FK_CodDisco_Musica Foreign Key (Cod_Disco) References USUARIO_AGENDA (Cod_Disco));	Criando a tabela MUSICA
ALTER TABLE MUSICA ADD NOM_AUTOR VarChar(40)	Adiciona o campo Nom_Autor
DROP TABLE MUSICA;	Elimina a tabela MUSICA

Encerre o *Database Desktop* e retorne ao *Delphi*.

Capítulo IV

Trabalhando com o Menu

Qualquer projeto precisa de um menu, fica mais prático para o nosso usuário navegar dentro de um projeto quando este é limitado por um menu principal, iniciaremos o nosso projeto no *Delphi* com a criação do Menu Principal do Sistema.


Metendo a Mão na Massa


A partir deste ponto, nossa aula se transforma em receita de bolo, a única coisa que você precisa fazer e seguir as orientações passo a passo, no princípio pode parecer meio idiota, mas afinal o computador é uma máquina idiota. Bom, vamos então metendo a mão na massa.

- Ao iniciar o *Delphi*, foi criado automaticamente um novo projeto, vamos descartá-lo e iniciar um novo. Para tanto:
 1. **Lembre-se:** no capítulo anterior criamos o diretório que abrigará o sistema a ser desenvolvido - **C:\SISTEMA\CADDISCO** - aprendemos o que é o **Alias** e estruturamos as nossas tabelas, se algum destes conceitos ficaram dispersos eu lhe aconselho que retorne ao capítulo anterior
 2. Crie um novo projeto digitando **File** e **New Application**. (Responda negativamente quaisquer mensagem para gravar o projeto atual).

Criando a janela do menu

A janela do menu principal é bem simples, como você já deve ter visto em vários aplicativos o menu é o objeto que fica servindo de pano de fundo para toda a aplicação, todo o trabalho é realizado com o auxílio de suas chamadas, em conjunto com o menu teremos três formulários gerenciadores que daremos o nome de:

- ✓ **F_Menu** - Menu principal propriamente dito;
 - ✓ **F_Sobre** - A janela "Sobre o sistema..."; e
 - ✓ **F_Inicio** - Janela *Splash* que iniciará o nosso aplicativo.
- Vamos criar inicialmente o nosso menu principal:
 1. Clique no botão  (Main Menu), localizado na *Component Palette* na página *Standard*, e clique dentro do objeto **Form1** (não se preocupe com a posição, pois este objeto ficará invisível quando o aplicativo for executado).
 - Foi criado neste momento o objeto *MainMenu1* derivado da classe de objeto *TMainMenu*, a partir deste objeto vamos criar nosso menu:

2. Dê um duplo clique em cima do objeto, ou clique na propriedade **Items** da *Object Inspector* aparecerá o botão . Clique neste botão.
- Observe a tela de propriedades do *Object Inspector*, neste momento vou me conter em falar das mais significativas, mas futuramente retomaremos o assunto:

Caption - Define o nome do item de menu, quaisquer nomes são válidos, incluindo acentos, o caractere especial "&" deve ser colocado uma única vez, ele causa o sublinhado da letra, tornando-a uma letra (em conjunto com a tecla **Alt**) de acesso a opção.

Enabled - Define se o item está disponível ou não para o usuário.

Name - Nome interno do item (colocado automaticamente na escolha do **Caption**).

ShortCut - Combinação de teclas, para um rápido acesso ao item (além da letra escolhida com "&").

- Inserindo os itens iniciais:
 1. Digite "&Arquivo" na propriedade **Caption**, em seguida pressione a tecla Enter.
 2. Clique no novo espaço aberto, criado lateralmente, e digite "&Consulta" na propriedade **Caption**, em seguida pressione a tecla Enter.
 3. Proceda da mesma forma criando as opções: "&Relatório" e "Au&xílio".
 4. Clique na opção **Arquivo**, aparecerá um espaço vazio abaixo, clique neste espaço e digite "&Tabela" na propriedade **Caption**. Ao ser dado **Enter** o *Delphi* criará mais um espaço abaixo, digite "&Cadastro" na propriedade **Caption**.
 5. Abaixo do **Cadastro**, digite "-" (sinal de menos) na propriedade **Caption** (o *Delphi* criará uma barra de separação) e altere a propriedade **Enabled** para *False*.
 6. No novo espaço criado, após a barra, digite "&Sair" na propriedade **Caption** e altere a propriedade **ShortCut** para *Ctrl+X*.
 7. Clique na opção **Tabela**, clique com o botão direito do mouse, aparecerá um *menu pulldown*, clique na opção **Create Submenu**.
 8. Digite "&Categoria" na propriedade **Caption**.



- Complete os próximos itens de modo que o menu fique:

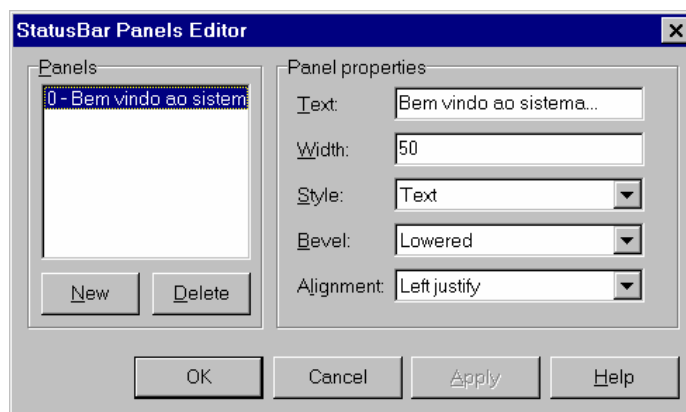
<u>A</u> rquivo	<u>C</u> onsulta	<u>R</u> elatório	Au <u>x</u> ílio
<u>T</u> abela ➤ <u>C</u> ategoria	<u>C</u> D's	<u>G</u> eral	<u>S</u> obre o sistema
<u>C</u> adastro	<u>M</u> úsica por CD's	<u>C</u> apa do CD	<u>C</u> onteúdo
<u>S</u> air Ctrl+X		<u>C</u> onfigura Impressora	<u>T</u> ópicos de Ajuda
			Como usar a <u>A</u> juda

- Saia da janela *Menu Designer* digitando Alt+F4, o menu já existe no objeto *form1*.
- Altere as seguintes propriedades para o objeto *form1*:

Propriedade	Valor	Descrição
BorderStyle	Single	Estilo da borda da janela; modo simples.
Caption	Compact Disc Digital Audio	Label escrito na tarja superior da janela.


Color	clMenu	Cor da janela, clMenu é uma constante que guarda a cor padrão da janela definido pelo usuário no Windows.
Name	F_Menu	Nome do objeto interno.
WindowState	wsMaximized	Modo de abertura da janela, modo Maximizado.

- Salvando o formulário e o projeto:
 1. Salve o Formulário nas opções de menu **File** e **Save** (ou pressione Ctrl+S), o *Delphi* questionará o nome e o diretório, o diretório (conforme criado no capítulo anterior) é o **C:\SISTEMA\CADDISCO** e para o nome digite **fMenu** (note que o nome externo e o mesmo do nome interno diferenciado por “_”, isto facilitará a identificação do formulário e da sua unidade).
 2. Salve o Projeto digitando **File** e **Save Project**, salve o projeto no diretório **C:\SISTEMA\CADDISCO** com o nome **CDDA**.
- Criando no menu uma linha de status:
 3. Clique no botão  (*StatusBar*) na página *Win95* da *Component Palette* e clique em qualquer posição do objeto *F_Menu*.
 4. Clique no objeto criado *StatusBar1* em seguida clique na propriedade *Panels*, para alterar esta propriedade clique no botão , aparecerá a janela da **Status Bar Panels Editor**, clique sobre o botão **New** e para a propriedade **Text** coloque “Bem vindo ao sistema...” e clique no botão OK.




5. Altere também a propriedade *Name* do objeto para *LinhaStatus*

U' Caso você esteja usando o *Delphi 1.0* crie a barra de status do seguinte modo:


1. Clique no botão  (*Panel*) na *Component Palette* na página *Standard* e clique em qualquer posição do objeto *F_Menu*.
2. Clique no objeto criado *StatusBar1* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
Align	alBottom	Alinhamento dentro do <i>form</i> , todo no rodapé
Alignment	taLeftJustify	Alinhamento da Caption , justificado à esquerda

BevelInner	bvLowered	Borda 3D interna, tipo pressionado
BevelOuter	bvLowered	Borda 3D externa, tipo pressionado
BorderWidth	1	Tamanho da borda
Caption	Bem vindo ao sistema...	Label do objeto
Name	LinhaStatus	Nome do objeto
Font	MS Sans Serif, Estilo da fonte: Normal, Tamanho: 8, Cor: Azul Marinho	Tipo de letra a ser mostrada no objeto, para alterar esta propriedade clique no botão 
Height	22	Altura do objeto

Inserindo os Códigos Iniciais

Vamos inserir o código para o objeto *LinhaStatus*, este objeto receberá os conteúdos da propriedade **hint** dos diversos objetos, formando assim uma linha de ajuda on-line na parte inferior do menu.


- Clique no botão  (*Toggle Form/Unit*) da *SpeedBar*, até você alternar para o *Code Editor*.

Abaixo da declaração: **Private** insira os códigos:


private	Procedures ou Funções Locais.
{ Private declarations }	
procedure ShowHint (Sender: TObject);	Cabeçalho de uma procedure Local.
public	Procedures ou Funções Públicas.
{ Public declarations }	
end;	Final da seção de declaração.

Abaixo da diretiva de compilação: **{\$R *.DFM}**


{\$R *.DFM}	Diretiva de compilação associando o nome do recurso externo ao mesmo nome do objeto <i>Form</i> .
procedure TF_Menu.ShowHint (Sender: TObject);	Cabeçalho da Procedure associado ao nome do objeto principal (TF_Menu).
begin	
LinhaStatus.Panels.Items[0].Text := Application.hint;	Atribui o valor do hint da aplicação ao Item criado do objeto <i>LinhaStatus</i> .
end;	

- Clique no botão  (*Toggle Form/Unit*) da *SpeedBar*, até você alternar para o *Form* clique no objeto *F_Menu* e na página *Events* da *Object Inspector*, dê um duplo clique no evento *OnCreate*.
 - O *Delphi* criou a procedure *FormCreate* a ser iniciada quando o objeto *F_Menu* for criado.
 - Digite o seguinte comando abaixo do comando **begin**:

procedure TF_Menu.FormCreate (Sender: TObject);	
begin	
Application.OnHint := ShowHint;	Atribui o valor da procedure ShowHint ao OnHint da aplicação.
end;	

- Clique no botão  (*Toggle Form/Unit*) da *SpeedBar*, até você alternar para o *Form* clique no objeto *MainMenu1* e entre no *Menu Designer*, e para cada opção de Menu altere as propriedades *hint* e *name* do seguinte modo:

Opção do Menu	Hint	Name
Arquivo	Cadastro e saída do sistema.	Arquivo1
Tabela	Informações básicas do sistema.	Tabela1
Categoria	Tipos de categoria para os CD's.	ItemTabela1
Cadastro	Inclusão e manutenção dos CD's.	Cadastro1
Sair	Saída do sistema e retorno ao Windows.	Sair1
Consulta	Verificação e pesquisa dos CD's cadastrados.	Consulta1
CD's	Localização dos CD's através de um filtro estabelecido.	ItemConsulta1
CD's por música	Localiza o CD através de um título de uma música.	ItemConsulta2
Relatório	Emissões em papel dos CD's cadastrados.	Relatorio1
Geral	Impressão dos CD's por um intervalo de código.	ItemRelatorio1
Capa do CD	Impressão de capas para os CD's.	ItemRelatorio2
Configura Impressora	Verifica a impressora a qual será destinado os relatórios.	ConfImpressora1
Auxílio	Formas de ajuda direta ao sistema.	Auxilio1
Sobre o sistema	Ajuda direta com o responsável pelo desenvolvimento.	ItemAuxilio1
Conteúdo	Manual On-Line direto.	ItemAuxilio2
Tópicos de Ajuda	Exibe os tópicos de ajuda do Manual On-Line.	ItemAuxilio3
Como usar a Ajuda	Mostra como utilizar o Auxílio On-Line.	ItemAuxilio4

- Saia do *Menu Designer*, salve o formulário e o projeto.
- Rode o projeto, clicando no botão  da *SpeedBar*, ou no menu principal a opção **R**un e **R**un, ou ainda, digite **F9**.
- Teste as opções do menu, veja na linha de Status os *hints* informados, saia com **Alt+F4**.

U' Caso você esteja usando o *Delphi 1.0* troque a procedure **ShowHint** para:

```
procedure TF_Menu.ShowHint (Sender: TObject);
```

```
begin
```

```
  LinhaStatus.Caption := Application.hint;
```

```
end;
```

Atribui o valor do hint da aplicação a Propriedade **Caption** do objeto *LinhaStatus*.

Iniciando os comandos do Menu

Com o menu pronto, começaremos a codificar os comandos que disponibilizamos ao nosso usuário:

- Automatizando o Comando **SAIR**: No objeto *F_Menu*, clique na opção **A**rquivo e clique na opção **S**air. O *Delphi* criou o evento Click para o objeto **Sair1**.
- Digite o seguinte comando abaixo do **begin**:

```
procedure TF_Menu.Sair1Click (Sender: TObject);
```

```
begin
```

```
  Close;
```

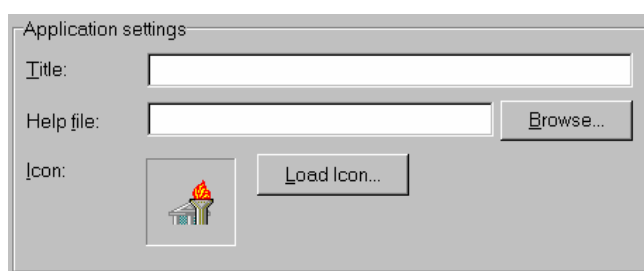
```
end;
```

proporciona o fechamento do formulário ativo

Colocando os comandos para o Auxílio

O *Delphi* implementa o auxílio on-line de maneira prática e eficiente, não ensinarei aqui como construir um arquivo .HLP, existem diversos aplicativos que já o fazem automaticamente e fica a seu critério o modo de criá-lo, apenas mostrarei como implementá-lo em seu projeto.

- Clique na opção **Project | Options...** e na página *Application* localize o seu arquivo com a opção **Help file**, aproveite também para nomear o projeto, com a opção **Title** e colocar um ícone para o projeto, opção **Icon** (clique no botão **Load Icon...**) e ao término clique no botão **OK**.



- Automatizando o Comando **CONTEÚDO**: No objeto *F_Menu*, clique na opção **Auxílio** e clique na opção **Conteúdo**. Digite:

```
procedure TF_Menu.ItemAuxilio2Click(Sender: TObject);
begin
  Application.HelpCommand(HELP_CONTENTS, 0);      Chama o arquivo de Ajuda
end;
```

- Automatizando o Comando **TÓPICOS DE AJUDA**: No objeto *F_Menu*, clique na opção **Auxílio** e clique na opção **Tópicos de Ajuda**. Digite:

```
procedure TF_Menu.ItemAuxilio3Click(Sender: TObject);
const
  EmptyString: pChar = "";                        Cria uma constante
begin
  Application.HelpCommand(HELP_PARTIALKEY, LongInt(EmptyString));  Tópicos do Ajuda
end;
```

- Automatizando o Comando **COMO USAR A AJUDA**: No objeto *F_Menu*, clique na opção **Auxílio** e clique na opção **Como usar a Ajuda**. Digite:

```
procedure TF_Menu.ItemAuxilio4Click(Sender: TObject);
begin
  Application.HelpCommand(HELP_HELPONHELP, 0);    Chama o auxílio do Windows
end;
```

- Quando fecharmos a nossa aplicação e necessário que também desativemos o auxílio, no objeto *F_Menu*, clique na página de **Events** e clique no evento **Destroy**. Digite:

```
procedure TF_Menu.FormDestroy(Sender: TObject);
begin
  Application.HelpCommand(HELP_QUIT, 0);          Desabilita o auxílio
end;
```


end;

- Saia do *Menu Designer*, salve o formulário e o projeto.
- Rode o projeto e teste as opções do menu, saia com **Ctrl+X** ou utilize o comando Sair.

U' Caso o **F1** não ative o auxílio On-Line, provavelmente o seu menu está com a propriedade *FormStyle* em modo **fsMDIForm**, coloque-a no modo **fsNormal**. Se mesmo assim ainda não funcionou, mude a propriedade *HelpContext* do formulário para **1**.

U' As palavras-chaves para o comando HelpCommand são:

Comando	Dados	Ação
HELP_CONTEXT	Inteiro longo, contendo o número do contexto.	Mostra o auxílio a partir de tópico selecionado identificado a partir do número do contexto definido pela seção [MAP] do arquivo .HPJ
HELP_CONTENTS	Ignorado. Normalmente passado 0.	Mostra o conteúdo do primeiro tópico definido pela seção [MAP] do arquivo .HPJ
HELP_SETCONTENTS	Inteiro longo, contendo o número do contexto que foi designado como tópico de conteúdo.	Determina a chamada do tópico determinado através do uso da tecla F1
HELP_CONTEXTPOPUP	Inteiro longo, contendo o número do contexto.	Mostra uma janela Pop-Up com um tópico particular indentificado pelo número do contexto definido pela seção [MAP] do arquivo .HPJ
HELP_KEY	Ponteiro longo como uma string contendo o tópico designado.	Mostra um tópico pesquisado em uma lista de palavras chaves. Esta palavra chave deve ser exatamente o texto procurado.
HELP_PARTIALKEY	Ponteiro longo como uma string contendo o tópico designado.	Mostra um determinado tópico através de uma lista de palavras chaves. Se a palavra chave não for encontrada posiciona na palavra fonética mais perto.
HELP_MULTIKEY	Ponteiro longo para uma estrutura de TMULTIKEYHELP. Esta estrutura específica de caracteres e palavras chaves.	Mostra o tópico indentificado pela palavra chave ou uma chave da tabela alternada.
HELP_COMMAND	Ponteiro longo, contendo a macro para a execução	Executa um macro help.
HELP_SETWINPOS	Ponteiro longo para uma estrutura de TMULTIKEYHELP. Esta estrutura específica contém o tamanho e a posição da janela do help primário ou a janela secundária para ser mostrado.	Mostra um help do windows com um mínimo de memória, variando o tamanho e a posição de acordo com o dado passado.
HELP_FORCEFILE	Ignorado. Normalmente passado 0.	Executa o WinHelp mostrando o arquivo de auxílio corrigido.
HELP_HELPONHELP	Ignorado. Normalmente passado 0.	Mostra o auxílio de como usar o auxílio.
HELP_QUIT	Ignorado. Normalmente passado 0.	Solicita o fechamento do auxílio ativo.

Criando a janela “Sobre o Sistema”

Em todo o sistema criado para o *Windows*® é incluído uma janela “Sobre o Sistema”, por uma boa razão! todos os outros sistemas para o *Windows*® possuem uma, e quem vai querer quebrar esta maravilhosa tradição e criar um sistema sem uma janela destas ?

Criar o primeiro formulário com o *Delphi* não é uma tarefa assim tão difícil, mas para perdemos o medo inicial vamos criar a *AboutBox* (ou CaixaSobre), observe:

Criando e alterando os objetos

- Para criar o formulário a partir do menu principal a opção **File** e **New...**, aparecerá a janela da *New Items* (mais informações retorne ao **Capítulo II**) e clique na página *Forms* e no objeto intitulado *About box*.
- Altere os seguintes objetos (localize-os através da *Object Inspector*):

AboutBox (classe TAboutBox):

Propriedade	Valor	Descrição
Caption	Sobre o sistema	Label escrito na tarja superior da janela
Name	F_Sobre	Nome interno do objeto

ProgramIcon (classe TImage):

Propriedade	Valor	Descrição
Picture	Escolha o BitMap de sua preferência	Objeto imagem

ProductName (classe TLabel):

Propriedade	Valor	Descrição
Caption	CDDA	Label do objeto
Font	Ms Sans Serif, Negrito Itálico, 18, Castanho	Tipo de letra a ser mostrada no objeto

Version (classe TLabel):

Propriedade	Valor	Descrição
Caption	Versão Beta Teste	Label do objeto
Font	Ms Sans Serif, Itálico, 8, Azul Marinho	Tipo de letra a ser mostrada no objeto

Copyright (classe TLabel):


Propriedade	Valor	Descrição
Caption	Copyright © 1995 - Nome da Empresa	Label do objeto
Font	Ms Sans Serif, Normal, 8, Azul marinho	Tipo de letra a ser mostrada no objeto

U' Para conseguir um © pressione ALT+184 e um ® pressione ALT+169.

Comments (classe TLabel):

Propriedade	Valor	Descrição
Caption	Compact Disc Digital Audio - Cadastro de CD's	Label do objeto

Font Ms Sans Serif, Normal, 8, Preto Tipo de letra a ser mostrada no objeto

- Elimine o objeto **OKButton** (botão de OK), clique sobre ele e pressione Delete, clique no objeto **F_Sobre** e click no objeto *BitBtn*  encontrado na *Component Palette* na página *Additional*, e click novamente no **F_Sobre**

Altere as seguintes propriedades:

Propriedade	Valor	Descrição
Kind	bkOK	Determina a classe a ser utilizada pelo botão, automaticamente será alterado as propriedades: Caption , Glyph e ModalResult
Hint	Retorna ao menu principal	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Name	ButOK	Nome do Objeto
Width	92	Largura do objeto

- Dê uma organizada geral quanto a posição dos objetos para você poder ter uma idéia compare o desenho do seu formulário para ver se não ficou faltando nada:



- Salve o formulário nas opções de menu **File** e **Save** (ou pressione Ctrl+S), o *Delphi* questionará o nome e o diretório, o diretório é o **C:\SISTEMA\CADDISCO** e para o nome digite **fSobre** (note que novamente o nome externo e o mesmo do nome interno diferenciado por “_”).

Associando o form “Sobre o Sistema” ao menu

Vamos associar o form **fSobre** com o **fMenu**:

- Automatizando o comando **SOBRE O SISTEMA**: No objeto *F_Menu*, clique na opção **Auxílio** e na opção **Sobre o Sistema**. O *Delphi* criou o evento **Click** para o objeto item do menu **ItemAuxilio1**.
- Digite o seguinte comando abaixo do **begin**:

```
procedure TF_Menu.ItemAuxilio1Click(Sender: TObject);  
begin  
    F_Sobre.ShowModal;           Abre o Objeto F_Sobre em modo Modal.  
end;
```

- Abaixo da diretiva de compilação: **{\$R *.DFM}**

*{\$R *.DFM}*

```
uses  
    fSobre;                     Utiliza a Unidade fSobre e todos os objetos dependentes dela.
```

```
procedure TF_Menu.ShowHint (Sender: TObject);
```

- Saia do *Code Editor* e salve o formulário e o projeto.
- Rode o projeto e teste o formulário Sobre o Sistema, o modo *Modal* não permitirá que você clique em nenhum outro lugar até a finalização desta janela, volte para o menu com o botão OK.

U' Note que não foi preciso colocar nenhum código para que ao pressionar o botão OK o formulário fosse fechado, isto foi realizado graças a opção **Kind**, no *Delphi* você encontrará outros modelos de botão padrão do tipo: Cancela, Sim, Não entre outros.

Criando a janela *Splash*

A janela *Splash* é tida como a mais importante da aplicação. Esta janela aparece uma única vez (no início) durante a execução do seu sistema informando ao usuário para ter paciência e aguardar tranqüilamente enquanto o sistema é carregado, os formulários são criados, etc.

Existem vários tipos de janela *Splash*. O tipo mais comum é aquela que mostra o nome da aplicação, o autor, a versão, direitos autorais (Copyright) e uma imagem ou ícone que identifica a aplicação. Através da característica de herança dos objetos vamos obter facilmente esta janela:

1. No menu principal selecione a opção **F**ile e **N**ew..., clique na página *CDDA* e na figura intitulada *F_Sobre*.

U' Neste momento você obteve uma cópia da janela *fSobre* o problema é que a janela *fSplash* terá menos objetos que a janela *fSobre*, e por característica de herança o filho sempre deve superar os pais nunca ao contrário, então é necessário que invertamos as duas janelas.

Modificando o objeto **F_Sobre**

1. Modifique a propriedade **Name** do *F_Sobre* para **F_Splash** e a propriedade **BorderStyle** para **bsNone**.
2. Clique sobre objeto *butOK* e digite Ctrl+X, acerte o tamanho da janela.

3. Salve o formulário nas opções de menu **File** e **Save As...**, o *Delphi* questionará o nome e o diretório, o diretório é o **C:\SISTEMA\CADDISCO** e para o nome digite *fSplash*.

Recriando o objeto **F_Sobre**

1. Clique no objeto *F_Sobre1* e note que automaticamente ele adquiriu as modificações do objeto **fSplash**.
2. Aumente o tamanho da janela de modo a que caiba novamente o botão de OK e digite CTRL+V, troque a propriedade **Caption** do *ButOK* para **&OK**.
3. Modifique a propriedade **BorderStyle** do *form* para **bsDialog** e a propriedade **Caption** para **Sobre o Sistema**.
4. Salve o formulário nas opções de menu **File** e **Save** (ou pressione CTRL+S), o *Delphi* questionará o nome e o diretório, o diretório é o **C:\SISTEMA\CADDISCO** e para o nome digite **fSobre** (confirme a operação de sobescrita).

Organizando o objeto **fSplash**

1. A partir do menu principal clique em **Project | Options...**, clique na página *Forms* e envie o objeto *F_Splash* (clique sobre ele e clique no botão com o sinal de >) que está na lista *Auto-create forms* para a lista *Available forms* (para não ocupar espaço em memória uma janela que só será utilizada uma única vez, criaremos esta janela via comandos), clique sobre o botão **OK**.
2. No menu principal selecione a opção **View** e **Project Source** (estamos agora acessando o programa principal que controla todas os outros formulários ou *units*).
3. Após o comando **begin** adicione as seguintes linhas:

<code>F_Splash := TF_Splash.Create(Application);</code>	Cria o form como parte da aplicação.
<code>F_Splash.Show;</code>	Chama o form de modo não modal.
<code>F_Splash.Refresh;</code>	Mostra o form e devolve o controle para a aplicação.

4. Antes do comando **Application.Run**; adicione a linha:

<code>F_Splash.Free;</code>	Libera o form da aplicação.
-----------------------------	-----------------------------

- Saia do *Code Editor* e salve o formulário e o projeto.
- Rode o projeto, qualquer problema compare com o código abaixo:

program CDDA;

uses

Fmenu **in** 'FMENU.PAS' {*F_Menu*},
FSobre **in** 'FSOBRE.PAS' {*F_Sobre*};

{*\$R *.RES*}

begin

F_Splash := TF_Splash.Create(Application);
F_Splash.Show;
F_Splash.Refresh;

```
Application.Initialize;  
Application.HelpFile := 'C:\Sistema\CadDisco\Guia.hlp';  
Application.CreateForm(TF_Menu, F_Menu);  
Application.CreateForm(TF_Sobre, F_Sobre);  
F_Splash.Free;  
Application.Run;  
end.
```

U' Você pode mover o comando **SplashScreen.Free;** para o evento **OnShow** do form *F_Menu*. Isto fará com que a janela **Splash** só desapareça quando o menu for ativado.

U' Infelizmente para os usuários do *Delphi 1.0* esta característica de herança não havia sido implementada então, faz-se necessário a construção da tela *fSplash* através de uma cópia da tela *fSobre* com a utilização do comando **Save As....**

Criando o acesso a Base de Dados


É bem verdade que o nosso sistema se encontra na base *Paradox*, mas como escrevi no começo do trabalho com o *Delphi* é possível modificar o repositório de dados sem precisar alterar uma só linha do sistema produzido.

No capítulo a respeito das tabelas também vimos a impossibilidade de algumas ações produzidas por cláusulas **SQL** não serem bem vindas em base de dados não totalmente compatíveis com a estrutura do **SQL**, então se faz necessário identificar uma *base padrão* de uma **base SQL**.

No menu principal insira o objeto *DataBase* , encontrado na *Component Palette* na página *Data Access*, e altere as seguintes propriedades:

Propriedade	Valor	Descrição
AliasName	AliasDisco	Nome do Sinônimo
DataBaseName	BaseDisco	Nome do banco de dados
Name	DBDisco	Nome do objeto

Insira agora os códigos que permitirá a abertura e o fechamento da base de dados:

- Clique no botão  (*Toggle Form/Unit*) da *SpeedBar*, até você alternar para a *Code Editor* e localize o procedimento *FormCreate* associado ao evento *OnCreate*:

```
procedure TF_Menu.FormCreate(Sender: TObject);
```

```
begin
```

```
Application.OnHint := ShowHint;
```

```
DBDisco.Connected := True;           Inicia o Banco de Dados
```

```
end;
```

A partir do próximo capítulo entraremos realmente no que o *Delphi* é capaz com tabelas, mas antes, é necessário que os conceitos ensinados anteriormente estejam bem fixados, se alguma coisa deu errada, releia o capítulo, ou então confira o código do *F_Menu*:

unit fMenu;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Menus, ComCtrls, DBTables, DB;

type

TF_Menu = **class**(TForm)
 MainMenu1: TMainMenu;
 Arquivo1: TMenuItem;
 Consulta1: TMenuItem;
 Relatorio1: TMenuItem;
 Auxilio1: TMenuItem;
 Tabela1: TMenuItem;
 Cadastro1: TMenuItem;
 N6: TMenuItem;
 Sair1: TMenuItem;
 ItemTabela1: TMenuItem;
 ItemConsulta1: TMenuItem;
 ItemConsulta2: TMenuItem;
 ItemRelatorio1: TMenuItem;
 ItemRelatorio2: TMenuItem;
 N12: TMenuItem;
 ConfImpressora1: TMenuItem;
 ItemAuxilio2: TMenuItem;
 ItemAuxilio3: TMenuItem;
 ItemAuxilio4: TMenuItem;
 N14: TMenuItem;
 ItemAuxilio1: TMenuItem;
 LinhaStatus: TStatusBar;
 DBDisco: TDatabase;
 procedure FormCreate(Sender: TObject);
 procedure Sair1Click(Sender: TObject);
 procedure ItemAuxilio2Click(Sender: TObject);
 procedure ItemAuxilio3Click(Sender: TObject);
 procedure ItemAuxilio4Click(Sender: TObject);
 procedure FormDestroy(Sender: TObject);
 procedure ItemAuxilio1Click(Sender: TObject);
 private
 procedure ShowHint (Sender: TObject);
 public
 { Public declarations }
end;

var

 F_Menu: TF_Menu;

implementation

{ \$R *.DFM }

uses

 fSobre;

procedure TF_Menu.ShowHint (Sender: TObject);

```

begin
  LinhaStatus.Panels.Items[0].Text := Application.hint;
end;

procedure TF_Menu.FormCreate(Sender: TObject);
begin
  Application.OnHint := ShowHint;
  DBDisco.Connected := True;
end;

procedure TF_Menu.Sair1Click(Sender: TObject);
begin
  DBDisco.Connected := False;
  Close;
end;

procedure TF_Menu.ItemAuxilio2Click(Sender: TObject);
begin
  Application.HelpCommand(HELP_CONTENTS, 0);
end;

procedure TF_Menu.ItemAuxilio3Click(Sender: TObject);
const
  EmptyString: pChar = '';
begin
  Application.HelpCommand(HELP_PARTIALKEY, LongInt(EmptyString));
end;

procedure TF_Menu.ItemAuxilio4Click(Sender: TObject);
begin
  Application.HelpCommand(HELP_HELPONHELP, 0);
end;

procedure TF_Menu.FormDestroy(Sender: TObject);
begin
  Application.HelpCommand(HELP_QUIT, 0);
end;

procedure TF_Menu.ItemAuxilio1Click(Sender: TObject);
begin
  F_Sobre.ShowModal;
end;

end.

```

U' Caso o seu sistema não seja migrado para nenhum banco de dados no padrão **SQL** (do tipo **ORACLE**[®], **SYBASE**[®], ...) não existe nenhuma necessidade em se utilizar o objeto **DataBase**, mas a utilização ou não do objeto não afeta o tempo de acesso ao sistema, então porque não prepará-lo para uma eventual mudança?

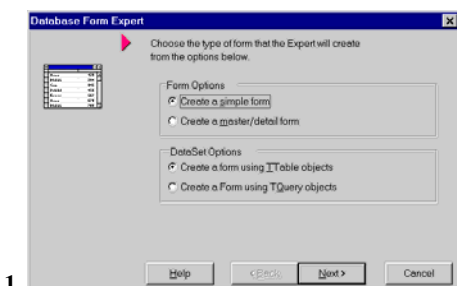
Capítulo V

Janela para as Tabelas

Tabelas primárias de informação requerem entradas de dados simples, no nosso caso temos a tabela de categoria, a criação de janelas para a sua manipulação de seus dados não é um bicho de sete cabeças como você verá a seguir.

Reabrindo o seu Projeto

- Se assim que você finalizou o capítulo anterior você saiu do *Delphi*, precisa agora reativar o projeto. Para tanto:
 - No menu principal clique em **F**ile e **O**pen....
 - O sistema desenvolvido é encontrado no diretório **C:\SISTEMA\CADDISCO**, com a extensão **.DPR**.
- Neste momento você está pronto para o trabalho, vamos criar a nossa janela:
 - Clique no menu principal a opção **F**ile e **N**ew..., em *New Items*, mude a página para *Forms* e clique no objeto entitulado *Database Form*, agora siga as telas:



1.

O tipo a ser criada.

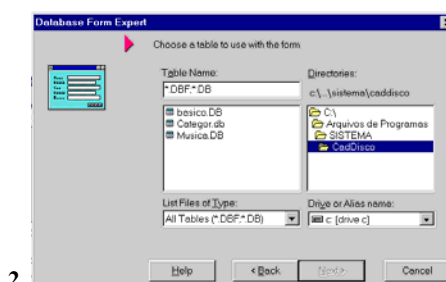
Form Options: Create a simple form

Uma janela simples

DataSet Options: Create a form using TTables objects

Usando o objeto tabela

Botão Next.



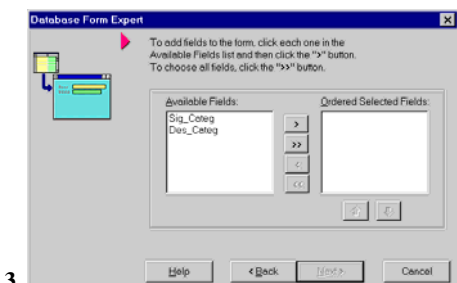
2.

A tabela a ser usada para a janela.

Drive or Alias name: AliasDisco

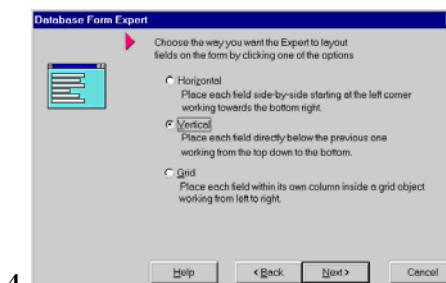
Table Name: categor.db

Botão Next



3.

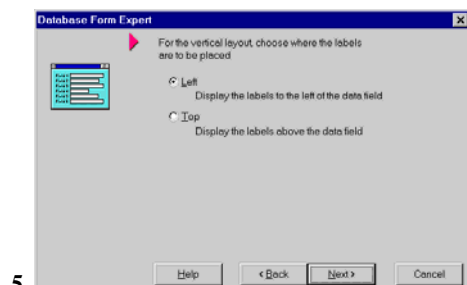
Campos a serem inseridos



4.

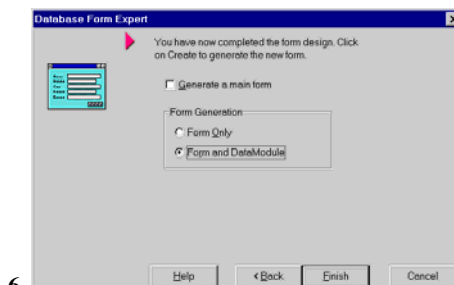
Formação dos campos

Botão ">>"
Botão Next



5. Posição dos Labels
Left - A esquerda
Botão Next

Vertical
Botão Next



6. Completo
Gera a tela como form principal - Não
O Que gerar: Form e DataModule
Botão Finish

Alterando as Janelas Criadas

Se você seguiu direito as orientações anteriores, então você está com um belo princípio de janela em suas mãos. Digo um belo princípio porque você há de concordar comigo que a janela gerada não é nenhum pouco amigável para o nosso usuário, programando há um certo tempo com o *Delphi* descobri um padrão de janela que meus usuários gostaram, mas você poderá futuramente também encontrar o seu próprio padrão de janela, então vamos a algumas alterações:

U' Se você está utilizando o *Delphi 1.0*, a única diferença será nos *DataModules*, não se preocupe coloque todas as instruções em um único formulário.

DataModules ?

Uma das principais novidades que acompanham o *Delphi 2.0* é a possibilidade de criação de *DataModules*, estas janelas especiais funcionam como uma espécie de repositório de dados, não são visualizáveis em tempo de execução. É possível colocar em um único *DataModule* todo o modelo relacional e todos os outros formulários do sistema acessando-o.

Acesse inicialmente o objeto *DataModule1* para as alterações que se seguem.


Modificando as Tabelas e as Ligações

Os objetos que contêm as **tabelas** e as **ligações de tabela** são objetos invisíveis quando o aplicativo está rodando portanto não se preocupe muito com a posição que ele ocupar.

Tabela no *Delphi* está contida no objeto  (*Table*), encontrado na *Component Palette* na página *Data Access*, este objeto não é a tabela em si, mas um ponteiro para a tabela, portanto

you poderá usar duplicações da mesma tabela, sem que isso afete a integridade de seu banco de dados.

- Alterando as propriedades da *Table*:
 1. Altere na propriedade **DataBaseName**, nome do banco de dados **BaseDisco**, caso você não encontre na lista o **BaseDisco**, abra o objeto *F_Menu* e tente novamente.
 2. Verifique se a propriedade **TableName** está apontada para a tabela: **CATEGOR.DB**, retire o **.DB** (visando a compatibilidade com outras bases)
 3. Coloque a propriedade **IndexFieldNames** no nome do índice primário da tabela. **SIG_CATEG**.
 4. Altere a propriedade **Name** para *TabCategor*, ou seja, Tab + Nome da tabela externo (sem a sua extensão), isto facilitará a identificação da *Table* e a qual *DataSource* que ela pertence..
 5. Outra propriedade interessante da *Table* é **Active**, ela define se a tabela está ou não ativada para o uso. Alteraremos esta propriedade via código, portanto não se preocupe muito com ela neste instante, o ideal é deixá-la *false*, i.é inativa.


As **Ligações da Tabela** no *Delphi* é realizada através do objeto  (*DataSource*), encontrado na *Component Palette* na página *Data Access*, este objeto faz a ligação de sua tabela externa com os campos do formulário.

- Alterando as propriedades do *DataSource*:
 1. Note que a propriedade **DataSet** está com o nome alterado (*TabCategor*), está propriedade define a *table* ou *query* (façamos nelas mais tarde) que será ligada.
 2. Altere a propriedade **Name** para **DsCategor**, ou seja, Ds + Nome da tabela externo (sem a sua extensão), como dito antes, isto facilitará a identificação do *DataSource* e a qual *Table* ele pertence.

Alterando os campos da tabela

Os campos da tabela deverão ser alterados para conterem as críticas, lembre-se, os campos presentes no *DataModule* são apenas uma “máscara” para os campos da tabela.

- Dê um duplo clique sobre o *TabCategor*, será aberta a janela do *FieldsEditor*.
- Marque o campo **SIG_CATEG** e altere as seguintes propriedades:

Propriedade	Valor	Descrição
DisplayLabel	Sigla	Nome do campo a ser mostrado na tela
Required	True	Se o campo é ou não requerido para inserção de dados na tabela
EditMask	>AA;0;_	Cria uma máscara de edição para o campo, para alterar esta propriedade clique no botão 

U'Outras propriedades importantes a serem levadas em consideração são:

- ✓ **Alignment**: Alinhamento dentro do campo: Centralizado, à esquerda ou à direita;

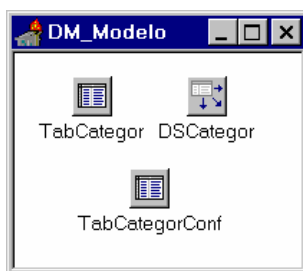
- ✓ **DisplayWidth**: Tamanho do campo disponível para inserção de dados;
- ✓ **FieldName**: Nome do campo na tabela, externo;
- ✓ **Name**: Nome do campo dentro do formulário, interno;
- ✓ **ReadOnly**: Se é um campo só de leitura;
- ✓ **Size**: Tamanho do campo na tabela; e
- ✓ **Visible**: Campo é ou não visível.

U'Veja mais observações sobre as máscaras no apêndice E.

- Marque o campo DES_CATEG e altere apenas a propriedade **DisplayLabel** para **Descrição**.


U'Troque a propriedade Name do objeto *DataModule1* para DM_Modelo.

Veja como ficou a visão final do objeto *DM_Modelo*, segue abaixo:



Codificando o DataModule

Você já deve ter notado que para o *Delphi* a escrita de códigos é bastante reduzida e bem dividida entre os eventos e com a criação dos *DataModules* o código ainda fica mais reduzido, diferentemente para os usuários de *Delphi 1.0*. Todas as críticas e controles para as tabelas ficarão no *DataModule* enquanto que o formulário se preocupará com o manuseamento dos campos.

- Código para efetivar as modificações na tabela para as bases SQL, clique no botão  (Toggle Form/Unit) da *SpeedBar*, até ter a visão novamente para o *DM_Modelo*, dê um clique simples no objeto *TabCategor* (marcando-o) e na *Object Inspector*, na página *Events*, dê um duplo click sobre o evento **AfterPost**:

```
procedure TDM_Modelo.TabCategorAfterPost(DataSet: TDataSet);
begin
  if F_Menu.DBDisco.IsSQLbased then           Se a base de dados é padrão SQL
  begin
    F_Menu.DBDisco.Commit;                   Gravando as alterações da tabela
    F_Menu.DBDisco.StartTransaction;         Reinicia o modo de transações
  end;
end;
```

U' O objeto que controla o banco de dados faz parte da Unit **fMenu** então é necessário fazer uso desta Unit, para tanto insira o seguinte código (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

```
uses
```

```
  fMenu;
```

U' Para as bases de formato SQL existem três declarações básicas:

1. StartTransaction - Inicia um bloco de controle para as declarações;
2. Commit - Termina o bloco de controle gravando as alterações feitas nas tabelas; e
3. RollBack - Termina o bloco de controle cancelando quaisquer modificações feitas nas tabelas.

Controlando a duplicação dos Campos-Chave

- É aconselhável não permitir que o usuário duplique os códigos de categoria, para tanto:
 1. Crie um novo objeto *Table* com as mesmas propriedades do objeto *TabCategor* (clique sobre o objeto e digite Ctrl+C e Ctrl+V, elimine as referências aos eventos), alterando a propriedade **Name** para **TabCategorConf**.
 2. No objeto *TabCategorSIG_CATEG*, localize-o através da *Object Inspector*, dê um duplo click sobre o evento **OnValidate**

```
procedure TDM_Modelo.TabCategorSig_CategValidate(Sender: TField);
```

```
begin
```

```
  if DSCategorState in [dsEdit, dsInsert] then      Verifica se o modo é de inserção ou edição de dados
```

```
  if TabCategorConf.FindKey([TabCategorSIG_CATEG]) then  Pesquisa o campo digitado na tab. criada
```

```
  begin
```

```
    F_Categ.EditSIG_CATEG.SetFocus;      Altera a posição do cursor para o objeto EditSIG_CATEG
```

```
    raise Exception.Create('Sigla da categoria duplicado'#10+      Caso já exista mostra
      'Click no botão "Localiza" em caso de dúvida');          mensagem de erro
```

```
  end;
```

```
end;
```

U' Note que existe uma referência para o objeto *F_Categ* (Será o formulário de Categoria) precisamos então fazer uso de sua Unit, para tanto coloque-a abaixo da diretiva de compilação:

```
{ $R *.DFM }
```

```
uses
```

```
  fMenu, { Menu Principal do Sistema }
```

```
  fCateg; { Cadastro de Categorias }
```

U' A declaração #10, funciona como um *Enter* dentro da mensagem, isto fará com que esta mensagem tenha duas linhas.

U' O segundo objeto **Table** foi criado pois a primeira tabela estará em modo de edição ou inserção de registros e não poderá ser desposicionada para a verificação, então a verificação se o registro existe será feita neste segundo objeto.

O comando **raise** impede que o registro duplicado seja adicionado na tabela, no modo run-time este comando provocará um erro de classe *exception* que travará o projeto, não se preocupe, digite F9 e prossiga com os testes, quando o projeto for compilado e rodado através do **.EXE** o erro não travará o projeto mostrando somente a mensagem definida.

- Um último detalhe para o *DataModule* que temos que prever que a cada novo registro o cursor deve se posicionar no primeiro campo do registro, para o início da digitação:
 1. Marque o objeto *TabCategor*, e dê um duplo click sobre o evento **OnNewRecord**:

```
procedure TDM_Modelo.TabCategorNewRecord(DataSet: TDataSet);
begin
    F_Categ.EditSig_Categ.SetFocus;           Altera a posição do cursor para o objeto EditSig_Categ
end;
```

Finalizando o DataModule

Salve o *DataModule* com o nome de DMModelo. Confira o código completo para o *DataModule*:

```
unit DMModelo;

interface

uses
    SysUtils, Windows, Classes, Graphics, Controls, Forms, Dialogs, DB, DBTables;

type
    TDM_Modelo = class(TDataModule)
        TabCategorSig_Categ: TStringField;
        TabCategorDes_Categ: TStringField;
        DSCategor: TDataSource;
        TabCategor: TTable;
        TabCategorConf: TTable;
        StringField1: TStringField;
        StringField2: TStringField;
        procedure TabCategorAfterPost(DataSet: TDataSet);
        procedure TabCategorSig_CategValidate(Sender: TField);
        procedure TabCategorNewRecord(DataSet: TDataSet);
    private
        { private declarations }
    public
        { public declarations }
    end;

var
    DM_Modelo: TDM_Modelo;

implementation

{$R *.DFM}

uses
    fMenu, { Menu Principal do Sistema }
    fCateg; { Cadastro de Categorias }
```

```
procedure TDM_Modelo.TabCategorAfterPost(DataSet: TDataSet);
begin
  if F_Menu.DBDisco.IsSQLbased then
  begin
    F_Menu.DBDisco.Commit;
    F_Menu.DBDisco.StartTransaction;
  end;
end;

procedure TDM_Modelo.TabCategorSig_CategValidate(Sender: TField);
begin
  if DSCategor.State in [dsEdit, dsInsert] then
  if TabCategorConf.FindKey([TabCategorSIG_CATEG]) then
  begin
    F_Categ.EditSIG_CATEG.SetFocus;
    raise Exception.Create('Sigla da categoria duplicado'#10+
      'Click no botão "Localiza" em caso de dúvida');
  end;
end;

procedure TDM_Modelo.TabCategorNewRecord(DataSet: TDataSet);
begin
  F_Categ.EditSig_Categ.SetFocus;
end;

end.
```

Comandos e suas funções, por ordem de aparição:

Uses - Faz o uso de determinada unidade de procedimentos e biblioteca de funções.

[DataSource].**State** - Define o estado em que se encontra determinado objeto **DataSource**.

[Objeto].**SetFocus** - Posiciona o cursor no objeto definido.

raise - Cria um erro de classe *exception* que não permitirá que qualquer outra ação prossiga até a mesma ser resolvida.

Alterando o Formulário

Com o *DataModule* concluído vamos atacar a janela que será mostrada para o nosso usuário, chame objeto *Form2* (chame-o através da *Project Manager* - opção do menu **View | Project Manager**).

Antes de fazermos quaisquer modificação vamos inicialmente alterar o nome da janela, para tanto pressione a tecla *F11* (aparecerá a *Object Inspector* para o objeto *Form2*) altere a propriedade **Name** para **F_Categ**.


Outra modificação importante é trocar a referência do comando **Uses** abaixo da diretiva de compilação que estava referenciado ao antigo nome do objeto *DataModule (Unit1)*, troque-o para:

```
{ $R *.DFM }
```

uses

DMModelo; { Referencia ao DataModule }

Modificando os Labels e Campos

Os **Labels**, representados pelo objeto  (*Label*), encontrado na *Component Palette* na página *Standard*, são as etiquetas de cada campo que aparece a esquerda dos campos.


- Alterando as propriedades dos Labels:
 1. Altere a propriedade **Caption** de SIG_CATEG e DES_CATEG para “&Sigla:” e “&Descrição:”, respectivamente.
 2. Modifique a propriedade **Fonte** de ambos para MS Sans Serif, Negrito, 8 e Castanho; e a propriedade **AutoSize** para **True**.

U' Para selecionar simultaneamente vários objetos, marque o primeiro objeto, segure a tecla SHIFT e marque os demais.

U' Qualquer problema para dimensionar o tamanho de objetos use a tecla SHIFT + Setas.


U' Qualquer problema para acertar a posição de objetos use a tecla CTRL + Setas.

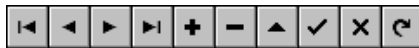
U' Uma propriedade interessante é a **FocusControl** ela indicará um controle para a posição do cursor. Ex.: Caso seja digitado ALT+S o cursor se posicionará no objeto **EditSig_Categ** ou Caso seja digitado ALT+D o cursor se posicionará no objeto **EditDes_Categ**.

Os **Campos de Edição**, representados pelo objeto  (*DBEdit*), encontrado na *Component Palette* na página *Data Controls*, são os que receberão o conteúdo dos campos da tabela.


- Alterando as propriedades do campo Código:
 1. Modifique a propriedade **Fonte** (de ambos os campos) para MS Sans Serif, Normal, 8 e azul marinho.
 2. Verifique as propriedades **DataSource** e **DataField**, nome da ligação com **DM_Modelo.DSCategor** e o nome do campo **Sig_Categ**, respectivamente.
 3. A propriedade **Name**, nome do campo, é montada com Edit + Nome externo do campo. **EditSig_Categ**.
- Alterando as propriedades do campo Descrição:
 4. Verifique as propriedades **DataSource** e **DataField**, nome da ligação com **DM_Modelo.DSCategor** e o nome do campo **Des_Categ**, respectivamente.
 5. A propriedade **Name**, nome do campo, é montada com Edit + Nome externo do campo. **EditDes_Categ**.

Objeto DBNavigator

O objeto para o controle da tabela, representada pelo objeto  (DBNavigator), encontrado na *Component Palette* na página *Data Controls*, e apresentada pôr uma barra de funções que ligada ao *DataSource* controla a navegação dos campos, adição de novos registros, edição e exclusão de registros, o cancelamento ou a confirmação de uma modificação e a atualização do banco de dados (quando em rede):



Apresentada pelos botões: nbFirst (primeiro), nbPrior (anterior), nbNext (próximo), nbLast (último), ndInsert (inserir), ndDelete (excluir), nbEdit (editar), nbPost (confirmar), nbCancel (cancelar) e nbRefresh (atualizar dados).

- Alterando as propriedades da barra de navegação:
 1. Confira a propriedade **DataSource** verificando para o nome da ligação com **DM_Modelo.DSCategor**
 2. A propriedade **ConfirmDelete** fará com seja exibida uma mensagem, confirmando ou não a exclusão.
 3. Altere a propriedade **Hints**, clicando em , digite o nome de cada botão do seguinte modo:

Primeiro
Anterior
Próximo
Último
Inserir
Excluir
Editar
Confirmar
Cancelar
Atualizar dados

4. Clique no botão OK e altere a propriedade **ShowHint** para **true**, isto fará com que embaixo de cada botão da barra, sobreposto pelo cursor, seja mostrado uma caixa como uma tarja amarela com a conteúdo da propriedade **Hint**.
5. Você poderá definir quais botões deverão aparecer na barra utilizando a propriedade **VisibleButtons**, para tanto clique no sinal de + que aparece a esquerda da opção e defina **true** ou **false** para os botões que serão ou não mostrados.

Modificando os Paineis

Existem dois objetos *Panel* criados automaticamente: o primeiro superior, abriga o objeto *DBNavigator*, o segundo ocupando o restante da janela, abriga um objeto do tipo *ScrollBox*, labels e campos.

- Alterando as propriedades do primeiro painel:

1. Altere a propriedade **Alignment**, alinhamento da propriedade **Caption** do painel, para **taLeftJustify**.
2. Coloque na propriedade **Caption** o nome **Categoria**.
3. Modifique a propriedade **Fonte** para MS Sans Serif, Itálico, 14 e azul marinho.

U' Arrume a barra de navegação de modo que não cubra a descrição, se for o caso aumente o tamanho da janela.

- Alterando as propriedades do segundo painel:
 1. Aumente ou diminua o tamanho da janela e note que este painel diminui e aumenta com ela, isto se deve a propriedade **Align** estar no modo **alClient**, altere esta propriedade (temporariamente) para **alNone**.

U' Para conseguir ver a tela de propriedades deste painel, clique em quaisquer das bordas, pois sobre este painel existe um outro objeto denominado *ScrollBar* também no modo **alClient**.

Modificando a Janela


A janela está um pouco escondida atrás dos objetos painéis criados, após alterar a propriedade **Align** do segundo painel, estique um pouco a janela para baixo e clique nela.

- Alterando as propriedades da janela:
 1. Retire as opções **biMinimize** e **biMaximize** da propriedade **BorderIcons**.
 2. Altere na propriedade **BorderStyle** para **bsSingle**.
 3. Mude a propriedade **Caption** para **Tabela**.
 4. Se você não o fez, mude a propriedade **Name** para **F_Categ**, ou seja **F_** + Nome da tabela externo (sem a sua extensão), isto facilitará a identificação entre o *form* e sua *unit*.
 5. Verifique a propriedade **Position** (posição da janela) ela deve estar com o valor **ScreenCenter** (centralizado).
 6. Salve o formulário com o nome **fCateg** e salve o projeto.

Criando o terceiro Painel

Criaremos agora um terceiro painel para comportar alguns botões

Criando e alterando o painel:


1. Crie um objeto *Panel*, clique no objeto  (*Panel*), encontrado na *Component Palette* na página *Standard*, e clique no objeto *F_Categ*, altere as seguintes propriedades:


Propriedade	Valor	Descrição
-------------	-------	-----------

Align	alBottom	Alinhamento dentro do <i>form</i> , todo no rodapé
Caption		Label do objeto
Height	41	Altura do objeto

Os botões que criaremos farão duas funções: 1.Sair da janela e 2.Localização rápida de um determinado registro.

Criando o primeiro botão:

1. Marque o objeto **panel3** criado, click no objeto *BitBtn* , encontrado na *Component Palette* na página *Additional*, e click no **panel3**.
2. Altere as seguintes propriedades para o botão:

Propriedade	Valor	Descrição
Kind	bkClose	Determina a classe a ser utilizada pelo botão, automaticamente será alterado as propriedades: Caption , Glyph e ModalResult
Caption	&Fechar	Label do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto, para alterar esta propriedade clique no botão 
Height	25	Altura
Hint	Retorna ao menu principal	Ajuda on-line para o objeto específico
Left	312	Alinhamento a esquerda
Name	ButFechar	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Width	89	Tamanho do objeto

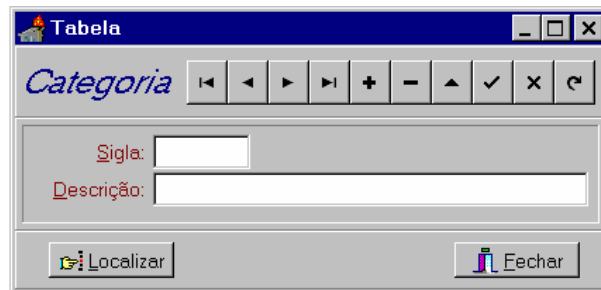
Criando o segundo botão:

1. Crie um segundo botão, alterando as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	&Localizar	Label do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\FIND.BMP	Imagem a ser mostrada no objeto
Height	25	Altura do objeto
Hint	Localiza determinado registro na tabela	Ajuda on-line para o objeto específico
Left	24	Alinhamento a esquerda
Name	ButLocalizar	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Width	89	Tamanho do objeto

- Alterando novamente as propriedades do segundo painel:
 1. Recoloque a propriedade **Align** do objeto *Panel2* para **alClient** e Salve o form e o projeto.

- Seu trabalho final deve ter ficado deste modo:



Programando no formulário

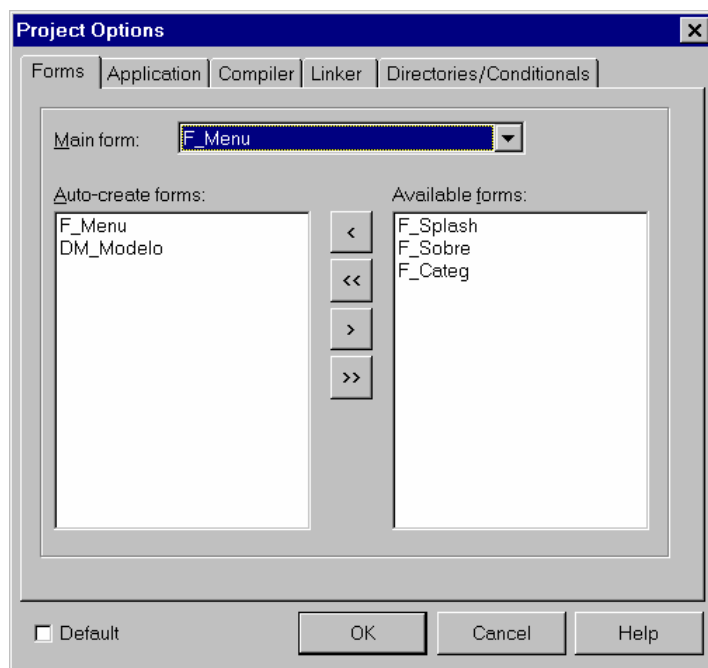
Agora vem a parte de código. Ao final deste tópico você observará que o trabalho maior ficou pôr conta de organizar e arrumar os objetos do que com o código em si, i.e., orientação a objetos.

Preservando as áreas de Memória

Com o *Delphi* é possível trabalhar de duas maneiras, a primeira é permitir que o *Delphi* crie todos os objetos em memória aguardando simplesmente que estes sejam chamados, mas isto implica que a máquina Client deva ter uma boa quantidade de memória para suportar os objetos que serão ali colocados, a segunda maneira e criarmos estes objetos via código permitindo que o *Delphi* crie o mínimo possível, o problema que isto implica é na demora quanto da chamada de um formulário, em média 50 segundos para ativar o formulário¹. É preferível trabalhar com o segundo modo uma vez que para entrar em determinado formulário o nosso usuário só o fará uma única vez.

Inicialmente retiraremos da área de criação automática o formulário de categoria e o formulário Sobre o sistema, para tanto, a partir do menu principal clique em **Project | Options...**, clique na página *Forms* e envie no objetos *F_Categ* e *F_Sobre* (clique sobre o primeiro, segure a tecla **Shift** e clique no segundo e em seguida clique no botão com o sinal de >) que está na lista *Auto-create forms* para a lista *Available forms*, clique sobre o botão **OK** (processo semelhante foi realizado para o formulário **F_Splash**).

¹ Este tempo é com base em uma máquina 486 DX2 com 8 Mb de memória.




- Altere agora a instrução do formulário **F_Menu**, evento **OnClick** para o objeto **ItemAuxilio1**, para criarmos o objeto **F_Sobre** e após a sua chamada destruí-lo da área de memória:

```

procedure TF_Menu.ItemAuxilio1Click(Sender: TObject);
begin
  with TF_Sobre.Create(Self) do           Cria o formulário em memória
  begin
    ShowModal;                             Chama o formulário através da área aberta
    Free;                                   Libera a área aberta
  end;
end;

```

- Código para ativar a Base de dados e as tabelas quando no *DataModule*, retorne ao formulário **F_Categ**:

1. Clique no botão  (*Toggle Form/Unit*) da *SpeedBar*, até você alternar para o *Code Editor*.

```

procedure FormCreate(Sender: TObject);
private
  { private declarations }
public
  { public declarations }
  procedure inicio;                       Criando a chamada para um procedimento público.
end;

var
  F_Categ: TF_Categ;

```

implementation

```
{ $R *.DFM }
```

uses

```
FMenu,      { Menu Principal do Sistema }
DMModelo;   { Referencia ao DataModule }
```

```
procedure TF_Categ.Inicio;
```

Início do procedimento

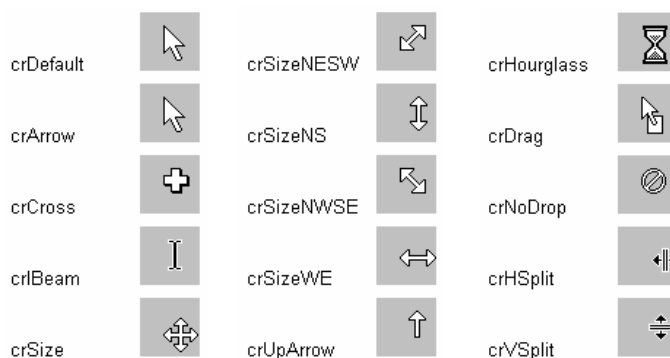
begin

```
  if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.StartTransaction;
  DM_Modelo.TabCategor.Open;
  Screen.Cursor := crDefault;
  ShowModal;
```

Se a base de dados é padrão SQL
Inicia o modo de transações
Ativa a Tabela
Faz o cursor ficar no formato de Seta
Mostra o formulário **F_Categ**

```
end;
```

U' A propriedade **cursor** no exemplo foi atribuída a unidade **Screen** que atribui ao sistema o modelo do cursor, mas é possível também atribuir um determinado cursor a um objeto específico, a propriedade pode ser alterada para os diversos tipos de cursores default do windows com as figuras que se seguem:



- Não é necessário colocar o comando **Close** para o objeto **butFechar** pois a propriedade **Kind** fará isto automático.
- Código para encerrar as tabelas do *DataModule* quando for dada saída no formulário, observe que o usuário não deve poder estar inserindo ou editando registros:
 1. No objeto *F_Categ*, localize-o através da *Object Inspector*, dê um duplo click sobre o evento **OnClose**:

```
procedure TF_Categ.FormClose(Sender: TObject; var Action: TCloseAction);
```

begin

```
  if DM_Modelo.DSCategor.State in [dsEdit, dsInsert] then
    begin
      MessageDlg('Cancele a edição (ou inserção) da Categoria antes de fechar!',
        mtInformation, [mbOK], 0);
```

Verifica se o estado do objeto
DataSource é Edição ou inserção
mostra mensagem
de informação

```
    Action := caNone;
```

Cancela a saída da janela

```
    Exit;
```

Sai da procedure

end;

```
Screen.Cursor := crHourGlass;
```

Faz o cursor virar uma ampulheta

TabCateg.Close;	Fecha a tabela
if F_Menu.DBDisco.IsSQLbased then	Se a base de dados é padrão SQL
F_Menu.DBDisco.Commit;	Encerra o modo de transações gravando
end;	as alterações no banco de dados

U' A função **MessageDlg** faz parte da Unit **Dialogs** então é necessário fazer uso desta Unit, para tanto insira o seguinte código (abaixo da diretiva de compilação):

{ \$R *.DFM }

uses

FMenu,	{ Menu Principal do Sistema }
DMModelo,	{ Referencia ao DataModule }
Dialogs;	{ Utilizado para o controle da função MessageDlg }

- Código para localizar determinado registro, observe que se o usuário não deve estar inserindo ou editando registros:

1. Dê um duplo Click sob o objeto *ButLocalizar*:

```

procedure TF_Categ.ButLocalizarClick(Sender: TObject);
var                                     Declaração de variáveis
  ObjPesquisa: String;                 Cria a variável ObjPesquisa do tipo String
begin
  if DM_Modelo.DSCategor.State in [dsEdit, dsInsert] then
    begin
      MessageDlg('Cancele a edição (ou inserção) da Categoria antes de localizar!',
        mtInformation, [mbOK], 0);
      Exit;
    end;
    ObjPesquisa := DM_Modelo.TabCategSig_Categ.Value;  Atribui a ObjPesquisa o valor do campo
                                                         de tabela Sig_Categ

    if InputQuery('Entre com a sigla da categoria',
      'Sigla',ObjPesquisa) then                               Solicita a digitação do código a ser procurado
    if not DM_Modelo.TabCateg.FindKey([ObjPesquisa]) then      Pesquisa o campo digitado na tabela
      MessageDlg('Sigla da Categoria não encontrada.',          Caso não seja encontrado informa
        mtInformation, [mbOK], 0);
end;

```

U' A função **InputQuery** também faz parte da Unit **Dialogs**.

U' O método **FindKey** faz parte do Objeto **TTable**.

Criando Funções Globais

Uma função ou um procedimento global e uma série de comandos comuns a um sistema como um todo, em linguagem Pascal é quase que proibido (não é proibido pois a linguagem permite) a utilização de um mesmo conjunto de comandos repetidas vezes, note que para o nosso formulário temos os mesmos comandos em chamadas diferentes:

```

procedure TF_Categ.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if DM_Modelo.DSCategor.State in [dsEdit, dsInsert] then           Aqui
    begin
      MessageDlg('Cancele a edição (ou inserção) da Categoria antes de fechar!',  Aqui

```

```

        mtInformation, [mbOK], 0);
    Action := caNone;
    Exit;
end;
...
end;

```

```

procedure TF_Categ.ButLocalizarClick(Sender: TObject);
var
    ObjPesquisa: String;
begin
    if DM_Modelo.DSCategor.State in [dsEdit, dsInsert] then
        begin
            MessageDlg('Cancele a edição (ou inserção) da Categoria antes de localizar!',
                mtInformation, [mbOK], 0);
            Exit;
        end;
    ...
end;

```

Podemos então retirar o trecho e criarmos uma função isolada que criticará o estado da edição devolvendo a mensagem, modificando o trecho diferente, ficando desta maneira (não esqueça de declarar a função na área PRIVATE):

```

private
    function CriticaEdicao(AntesDe: String) : boolean;
public
    procedure inicio;
end;

var
    ...
    ...
    ...

function TF_Categ.CriticaEdicao(AntesDe: String) : boolean;
begin
    if DM_Modelo.DSCategor.State in [dsEdit, dsInsert] then
        begin
            MessageDlg('Cancele a edição (ou inclusão) da categoria antes de ' +
                AntesDe, mtError, [mbOK], 0);
            Result := True;
        end
    else
        Result := False;
    end;

procedure TF_Categ.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    if CriticaEdicao('fechar') then
        ...
    end;
    ...
end;

procedure TF_Categ.ButLocalizarClick(Sender: TObject);

```



```

var
  ObjPesquisa: String;
begin
  if CriticaEdicao('localizar' ) then                Substitui pela chamada a função
    Exit;
  ...
end;

```

Note que já ganhamos uma certa vantagem, ao invés de termos que alterar em dois lugares diferentes só teremos que alterar em um único lugar, mas ainda não está perfeito pois devemos lembrar que um sistema normalmente não é composto por apenas uma tabela, sem contar a parte do cadastro, então se seguirmos o mesmo padrão de construção de formulários para outras tabelas continuaremos a repetir vários comandos, então vamos fazer que a nossa função sirva para a crítica de edição de qualquer tabela, para isto precisamos enviar também o *DataSource* que pesquisará o estado e uma outra variável do tipo *String* para dizermos de qual tabela estamos falando para cancelar a edição, vá para o objeto *F_Menu* e crie a seguinte função (não esqueça de declarar na área *PUBLIC*):

```

private
  procedure ShowHint (Sender: TObject);
public
  function CriticaEdicao(DSOrigem: TDataSource; DoQue, AntesDe: String) : boolean;    Aqui
end;

```

```

var
...
...
{ Função Crítica Edição
Recebe: DSOrigem: DataSouce para investigar o estado
      DoQue: Nome real da Tabela
      AntesDe: Função a executar do tipo Fechar, Localizar...
Devolve: True - Se o DataSource está em estado de edição ou inserção
        False - Se o DataSource está em estado de navegação }

```

```

function TF_Menu.CriticaEdicao(DSOrigem: TDataSource; DoQue, AntesDe: String) : boolean;
begin
  if DSOrigem.State in [dsEdit, dsInsert] then
    begin
      MessageDlg('Cancele a edição (ou inclusão) ' + DoQue + ' antes de ' +
        AntesDe, mtError, [mbOK], 0);
      Result := True;
    end
  else
    Result := False;
end;

```

Altere agora o objeto *F_Categ* eliminando a função **CriticaEdicao** e modificando as chamadas:

```

private
  { comandos particulares }      Elimine
public
  procedure inicio;
end;

var
  F_Categ: TF_Categ;

```

```

...
...
Elimine as linhas da função
...
...
procedure TF_Categ.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if F_Menu.CriticaEdicao(DM_Modelo.DSCategor, 'Categoria', 'fechar' ) then   Substitua aqui
  begin
    Action := caNone;
    Exit;
  end;
  ...
end;

procedure TF_Categ.ButLocalizarClick(Sender: TObject);
var
  ObjPesquisa: String;
begin
  if F_Menu.CriticaEdicao(DM_Modelo.DSCategor, 'Categoria', localizar' ) then   Substitua aqui
  begin
    Exit;
  end;
  ...
end;

```

Alterando o Menu para receber o formulário

Agora finalmente vamos rodar nosso projeto mas antes precisamos chamar o formulário através do menu principal para isto:

1. Abra o objeto **F_Menu**: no menu principal escolha **View** e **Project Manager**, marque o objeto *F_Menu* e clique sobre o botão **View form**.
2. Clique em **A**rquivo, **T**abela e **C**ategoria, coloque o seguinte código:

```

procedure TF_Menu.CategoriaClick(Sender: TObject);
begin
  Screen.Cursor := crHourGlass;           Faz do cursor uma ampulheta
  F_Categ := TF_Categ.Create(Application);  Cria o formulário em memória
  F_Categ.Inicio;                          Chama o formulário através da área aberta
  F_Categ.Free;                            Libera a área aberta
  Screen.Cursor := crDefault;             Faz do cursor uma seta
end;

```

3. O objeto *F_Categ* faz parte da Unit **fCateg** então é necessário fazer o uso desta Unit, para tanto insira o seguinte código (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

```
fSobre, { Janela do Sobre o Sistema }
fCateg; { Cadastro da Tabela de Categoria }
```

4. Saia do *Code Editor* e salve o formulário e o projeto.
5. Rode o projeto e teste o formulário de categoria, insira algumas categorias, tente provocar o erro de duplicação, tente inserir um registro com o código vazio e localizar um registro.

6. Se alguma coisa deu errada, releia o capítulo, ou então confira todo o código:

unit fcateg;

interface

uses

SysUtils, Windows, Messages, Classes, Graphics, Controls,
StdCtrls, Forms, DBCtrls, DB, Mask, ExtCtrls, Buttons;

type

TF_Categ = class(TForm)
 ScrollBar: TScrollBar;
 Label1: TLabel;
 EditSig_Categ: TDBEdit;
 Label2: TLabel;
 EditDes_Categ: TDBEdit;
 DBNavigator: TDBNavigator;
 Panel1: TPanel;
 Panel2: TPanel;
 Panel3: TPanel;
 ButFechar: TBitBtn;
 ButLocalizar: TBitBtn;
 procedure FormClose(Sender: TObject; **var** Action: TCloseAction);
 procedure ButLocalizarClick(Sender: TObject);

private

{ private declarations }

public

procedure inicio;

end;

var

F_Categ: TF_Categ;

implementation

{ \$R *.DFM }

uses

fMenu, { Menu Principal do Sistema }
DMModelo, { Referencia ao DataModule }
Dialogs; { Utilizado para o controle da função MessageDlg }

procedure TF_Categ.Inicio;

begin

if F_Menu.DBDisco.IsSQLbased **then**
 F_Menu.DBDisco.StartTransaction;
 DM_Modelo.TabCategor.Open;
 Screen.Cursor := crDefault;
 ShowModal;

end;

procedure TF_Categ.FormClose(Sender: TObject; **var** Action: TCloseAction);

begin

if F_Menu.CriticaEdicao(DM_Modelo.DSCategor, 'Categoria', 'Fechar') **then**
 begin

```
    Action := caNone;
    Exit;
end;
Screen.Cursor := crHourGlass;
DM_Modelo.TabCategor.Close;
if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.Commit;
end;

procedure TF_Categ.ButLocalizarClick(Sender: TObject);
var
    ObjPesquisa: String;
begin
    if F_Menu.CriticaEdicao(DM_Modelo.DSCategor, 'Categoria', 'Localizar' ) then
        Exit;

    ObjPesquisa := DM_Modelo.TabCategorSig_Categ.Value;
    if InputQuery('Entre com a Sigla da categoria','Sigla',ObjPesquisa) then
        if not DM_Modelo.TabCategor.FindKey([ObjPesquisa]) then
            MessageDlg('Sigla da Categoria não encontrada.',mtInformation,[mbOK],0);
end;

end.
```

Comandos e suas funções, por ordem de aparição:

[Tabela].**Open** - Ativa um objeto **Table** é equivalente a **Active := True**.

[Form].**ShowModal** - Ativa o objeto **Form**, não permitindo que nenhum outro objeto **Form** anterior seja ativado, até que o mesmo seja desativado.

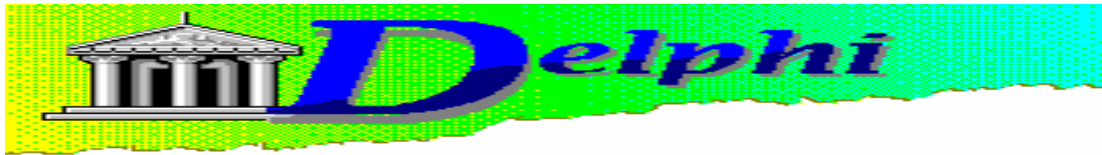
Exit - Sai da função ou procedimento.

[Tabela].**Close** - Desativa um objeto **Table** é equivalente a **Active := False**.

Var - Define uma cadeia de variáveis locais.

InputQuery - Mostra uma caixa de diálogo para leitura e entrada de variáveis do tipo **String**.

[Tabela].**FindKey** - Função que realiza uma pesquisa indexada no objeto **Table**, através do índice definido, retorna *True* se encontrou ou *False* se fim de arquivo



Capítulo VI

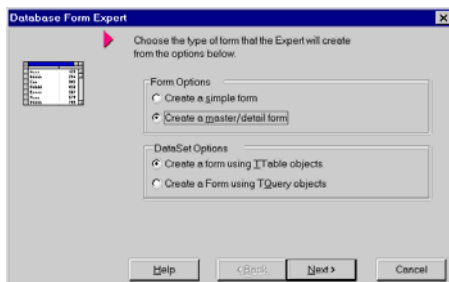
Trabalhando com janela Pai X Filha

Se você até agora não sentiu dificuldade em criar e entender o trabalho com tabelas livres, não sentirá também dificuldade em criar o formulário para receber este caso, ao contrário, aconselho que você releia e refaça o capítulo anterior.

Em nosso projeto, cada registro na tabela de **música** só existirá se houver um correspondente na tabela **básico**, então a tabela **básico** é “pai” (mestre) da tabela **música** que é sua “filha” (detalhe). Esta teoria acima é explicada no conceito de modelo relacional de dados (MER). O *Delphi* incorpora este modelo mesmo para banco de dados não-relacionais, caso estivéssemos utilizando o dBase em nosso projeto.

Criando a janela automaticamente

- Se assim que você finalizou o capítulo anterior você saiu do *Delphi*, reative o seu projeto.
- Agora que você está pronto para o trabalho, vamos criar a nossa janela:
 1. Clique no menu principal a opção **File** e **New...**, em *New Items*, mude a página para *Forms* e clique no objeto intitulado *Database Form*, parece cópia do capítulo anterior? cuidado leia as instruções abaixo:

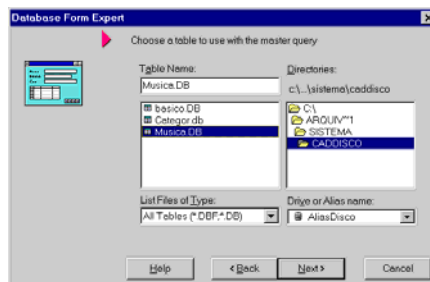


1.

O tipo a ser criada.

Form Options: Create a master/detail form
Uma janela mestre e detalhes

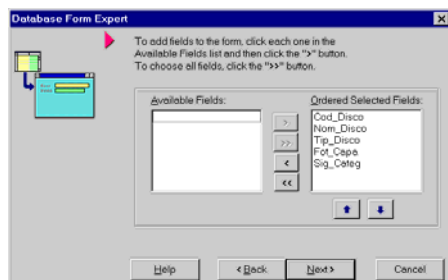
DataSet Options: Create a form using TTables objects
Usando o objeto tabela
Botão Next.



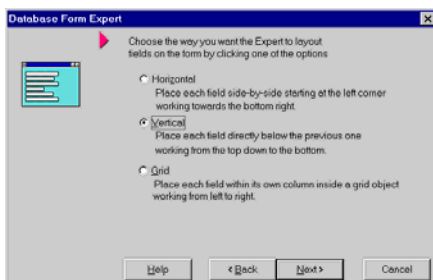
2.

A tabela mestre a ser usada para a janela.

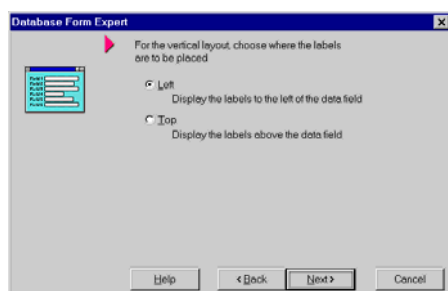
Drive or Alias name: AliasDisco
Table Name: basico.dbf
Botão Next



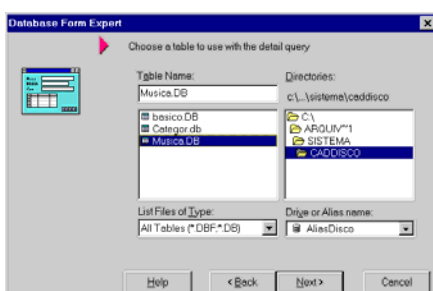
3. Campos a serem inseridos
Botão “>>”
Botão Next



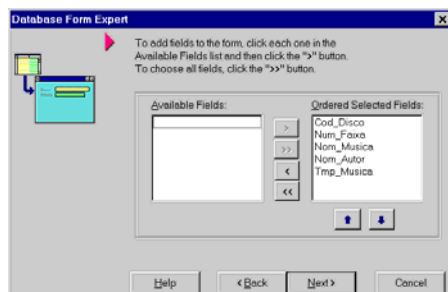
4. Formação dos campos
Vertical
Botão Next



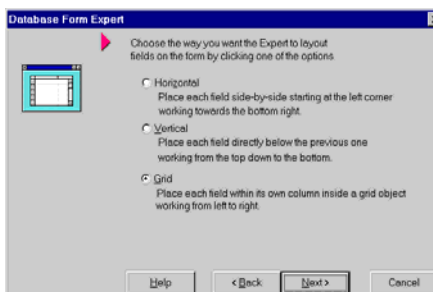
5. Posição dos Labels
Left - A esquerda
Botão Next



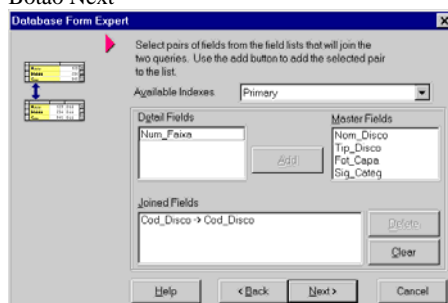
6. A tabela detalhe a ser usada para a janela.
Drive or Alias name: AliasDisco
Table Name: musica.dbf
Botão Next



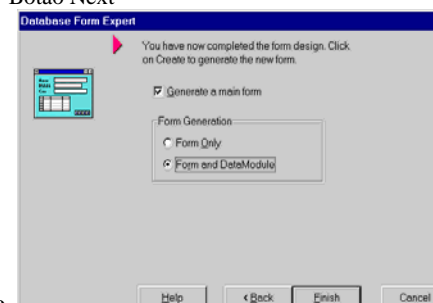
7. Campos a serem inseridos
Botão “>>”
Botão Next



8. Formação dos campos
Grid
Botão Next



- 9.



- 10.

Montagem da chave de ligação

Available Indexes : Primary

Detail Fields : COD_DISCO

Master Fields : COD_DISCO

Botão Add

Joined Fields COD_DISCO -> COD_DISCO

Botão Next

Completo

Gera a tela como form principal - Não

O Que gerar: Form e DataModule



Botão Finish

Sobre os DataModules

Como eu disse no capítulo anterior é possível criar um único *DataModule* abrangendo o modelo relacional completo, basta para isto você fazer o formulário que está chamando o *DataModule* controlar o comando *Open* e *Close* das tabelas. Não farei desta maneira pois isto ao mesmo tempo que simplificaria o meu trabalho dificultaria o seu entendimento, que é o de uma pessoa que estivesse aprendendo o *Delphi* neste momento, então para este trabalho adotarei um *DataModule* para cada cadastro.

Trabalhando com as Tabelas

Chame o objeto *DataModule1* criado, a nossa primeira providência será a de alterar a propriedade **Name** do objeto para **DM_Basico**, observe que foi criado dois objetos *Table*, o primeiro está apontado para a tabela BASICO e o segundo para a tabela MUSICA, note que para este segundo as propriedades **MasterSource** e **MasterFields** estão “presas” pelo primeiro objeto, este é o relacionamento entre ambas. Vamos antes criarmos alguns pequenos detalhes:

- Crie três objetos *Table* , encontrado na *Component Palette* página *Data Access*, e um objeto *DataSource* , encontrado na *Component Palette* página *Data Access*, e altere as seguintes propriedades:

Para o objeto Table1 (Já existente):

Propriedade	Valor	Descrição
DatabaseName	BaseDisco (se este valor não estiver disponível chame o objeto F_Menu)	Nome do Banco de Dados ou a localização do diretório das tabelas
TableName	BASICO	Nome externo da tabela
Name	TabBasico	Nome do objeto
IndexFieldNames	NOM_DISCO	Nome do campo indexado

Para o objeto DataSource1 (Já existente):

Propriedade	Valor	Descrição
DataSet	TabBasico	Nome da tabela vinculada
Name	DSBasico	Nome do objeto

Para o objeto Table2 (Já existente):

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Banco de Dados ou a localização do diretório das tabelas

TableName	MUSICA	Nome externo da tabela
Name	TabMusica	Nome do objeto
IndexFieldNames	COD_DISCO	Nome do campo indexado
MasterSource	DSBasico	Nome do DataSource Mestre
MasterFields	COD_DISCO	Campo de ligação da tabela Mestre

Para o objeto DataSource2 (Já existente):

Propriedade	Valor	Descrição
DataSet	TabMusica	Nome da tabela vinculada
Name	DSMusica	Nome do objeto

Para o objeto Table3 (Criado):

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Banco de Dados ou a localização do diretório das tabelas
TableName	BASICO	Nome externo da tabela
Name	TabBasicoConf	Nome do objeto
IndexFieldNames	NOM_DISCO	Nome do campo indexado
ReadOnly	True	Somente para leitura

Para o objeto Table4 (Criado):

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Banco de Dados ou a localização do diretório das tabelas
TableName	CATEGOR	Nome externo da tabela
Name	TabCategor	Nome interno da tabela
IndexFieldNames	SIG_CATEG	Nome do campo indexado
MasterSource	DSBasico	Nome do DataSource Mestre
MasterFields	SIG_CATEG	Campo de ligação da tabela Mestre
ReadOnly	True	Somente para leitura

Para o objeto DataSource3 (Criado):

Propriedade	Valor	Descrição
DataSet	TabCategor	Nome da tabela vinculada
Name	DSCategor	Nome do objeto


Para o objeto Table5 (Criado): Posicione-o ao lado do objeto *DSCateg*.

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Banco de Dados ou a localização do diretório das tabelas
TableName	CATEGOR	Nome externo da tabela
Name	TabCategorConf	Nome do objeto
IndexFieldNames	SIG_CATEG	Nome do campo indexado
ReadOnly	True	Somente para leitura

UOs objetos na hora da execução do formulário ficarão invisíveis, mas é bom colocá-los em cantos estratégicos, isto evita a confusão.

U' A utilidade de cada objeto *Table*:

- ✓ TabBasico - Tabela mestre principal.
- ✓ TabBasicoConf - Tabela BASICO para realizar a validação do nome do CD.
- ✓ TabMusica - Tabela utilizada para mostrar as músicas cadastradas de cada disco.
- ✓ TabCategor - Tabela para mostrar a descrição da categoria.
- ✓ TabCategorConf - Tabela para validação da sigla da categoria.

- Crie o objeto *Query* , encontrado na *Component Palette* página *Data Access*, que servirá para calcular o código automático, calculando sempre o código de maior valor, e altere as seguintes propriedades:

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Banco de Dados ou a localização do diretório das tabelas
Name	QryContador	Nome do Objeto
SQL	Select Max(COD_DISCO) from BASICO	Cláusula SQL, selecione o maior valor do campo COD_DISCO da tabela BASICO

U' O campo COD_DISCO criado, servirá apenas como uma chave de ligação entre a tabela Basico e Musica, será uma chave interna do nosso sistema e sua alimentação se fará através deste objeto SQL criado pegando o maior valor e adicionando 1.

Trabalhando com os Campos

Precisamos inserir alguns campos, pois iremos precisar deles no trabalho com o código e para o objeto *DBGrid1*, utilize o *Fields Editor* para inserir os campos que faltam:

Para o objeto TabBasico:

- ✓ COD_DISCO, Propriedade **DisplayLabel**: Código;
- ✓ NOM_DISCO, Propriedade **DisplayLabel**: Nome;
- ✓ TIP_DISCO, Propriedade **DisplayLabel**: Tipo e **EditMask**: >AAA;0;_
- ✓ FOT_CAPA, Propriedade **DisplayLabel**: Foto; e
- ✓ SIG_CATEG, Propriedade **DisplayLabel**: Categoria e **EditMask**: >AA;0;_

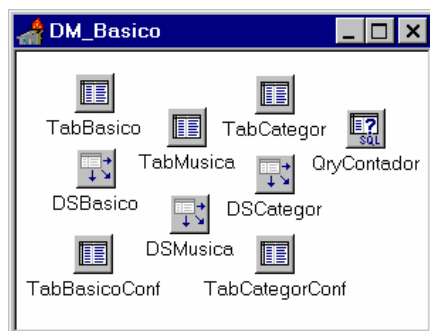
Para o objeto TabMusica:

- ✓ COD_DISCO, Propriedade **Visible**: False.
- ✓ NUM_FAIXA, Propriedade **DisplayLabel**: Faixa e **DisplayWidth**: 2;
- ✓ NOM_MUSICA, Propriedade **DisplayLabel**: Música e **DisplayWidth**: 40; e
- ✓ NOM_AUTOR, Propriedade **DisplayLabel**: Autor; e
- ✓ TMP_MUSICA, Propriedade **DisplayLabel**: Tempo e **EditMask**: 00\;00;0;_

Para o objeto QryContador:

- ✓ De um duplo clique sobre o objeto aparecerá a janela **Fields Editor**, clique com o botão direito e no menu que aparecerá clique na opção **Add Fields...** e adicione o campo.

Antes de iniciarmos a programação do *DataModule* compare como ficou o *DataModule* e salve-o com o nome de **DMBasico**:



Controlando o DataModule

Agora falta o código, note que a maior parte é uma repetição daquilo que já vimos anteriormente:

- Ativando as tabelas ao ser chamado *DataModule*:
 1. Clique no objeto *DMBasico* e na *Object Inspector* na página *Events* dê um duplo clique sobre o evento **OnCreate** e altere-o do seguinte modo:

```
procedure TDM_Basico.DM_BasicoCreate(Sender: TObject);
begin
  if F_Menu.DBDisco.IsSQLbased then           Se a base de dados é padrão SQL
    F_Menu.DBDisco.StartTransaction;         Inicia o modo de transações
  TabCategor.Open;                           Ativa as tabelas
  TabCategorConf.Open;
  TabMusica.Open;
  TabBasico.Open;
  TabBasicoConf.Open;
end;
```

- Código para encerrar as transações com a Base de Dados e fechar as tabelas quando for encerrado o *DataModule*.

1. Digite F11 e localize o objeto *DM_Categ*, na *Object Inspector*, na página *Events*, dê um duplo click sobre o evento **OnDestroy**:

```
procedure TDM_Basico.DM_BasicoDestroy(Sender: TObject);
begin
  TabBasico.Close;                           Fecha as tabelas
  TabBasicoConf.Close;
  TabMusica.Close;
  TabCategor.Close;
  TabCategorConf.Close;
  if F_Menu.DBDisco.IsSQLbased then           Se a base de dados é padrão SQL
    F_Menu.DBDisco.Commit;                   Encerra o modo de transações gravando as
                                              alterações no banco de dados
end;
```

U' Lembre-se da utilização do objeto **F_Menu**, então é preciso declarar a unidade a qual ele pertence com o comando **USES** (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

```
fMenu; { Menu Principal }
```

U' Repare que no início desta **Unit** também existe uma outra declaração **Uses**, após a sessão **interface**, então por que não colocar todas essas declarações em um lugar só? 1º) O *Delphi* controlará (colocando ou removendo) as Units ali colocadas (dependendo dos objetos utilizados) e 2º) Todos os comandos declarados antes da declaração **implementation** (com exceção de eventos de criação do tipo **onCreate**), serão executados e **objetos** e **units** ficarão em memória esperando serem chamados, então é impraticável colocar **units** que só serão utilizadas em tempo de execução.

Contadores

O objeto *query* realiza consultas em modo SQL, no próximo capítulo o utilizaremos para criarmos nossas consultas mas, neste momento ele será utilizado para verificar qual o maior valor armazenado no campo código.

- A cada novo registro devemos criar também um novo COD_DISCO (lembra do objeto *QryContador*). Além disso, precisamos nos posicionar no primeiro campo do cadastro, isto será realizado em dois eventos distintos:
 1. Marque o objeto *TabBasico*, e dê um duplo clique sobre o evento **onNewRecord**:

```
procedure TF_Basico.TabBasicoNewRecord(DataSet: TDataSet);
```

```
begin
```

```
    F_Basico.EditNOM_DISCO.SetFocus;
```

Posiciona o cursor no objeto

```
end;
```

U' Observe que foi utilizado o objeto **F_Basico**, então é preciso declarar a unidade a qual ele pertence com o comando **USES** (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

```
fBasico; { Cadastro do Básico }
```

```
fMenu; { Menu Principal }
```

2. Digite F11 e dê um duplo clique sobre o evento **BeforePost**:

```
procedure TF_Basico.TabBasicoBeforePost(DataSet: TDataSet);
```

```
begin
```

```
    if DSBasico.State = dsInsert then
```

Verifica se é uma inclusão na base

```
    begin
```

```
        QryContador.Active := False;
```

Desativa a Query

```
        QryContador.Active := True;
```

Ativa a Query

```
        with QryContador.Fields[0] do
```

Observe os lembretes

```
            if IsNull then
```

Se o valor do contador é nulo

```
                TabBasicoCOD_DISCO.Value := 1
```

Atribui 1 ao campo COD_DISCO

```
            else
```

Senão

```
                TabBasicoCOD_DISCO.Value := AsInteger + 1;
```

Atribui ao campo COD_DISCO a adição

```
end;                                de 1 ao valor do contador campo
end;
```

U' O comando **with** é utilizado como método de taquigrafia, para não escrevermos várias vezes o mesmo nome de um determinado objeto.

U' Porque não colocarmos todo o código no evento **onNewRecord** ? Por causa do controle multi-usuário, imagine, um indivíduo **A** inicia a inclusão de um CD, e um indivíduo **B** também inicia outra inclusão, como o código do indivíduo **A** ainda não foi gravado no banco, será dado o mesmo código para o indivíduo **B**, isto não acontecerá se o código for calculado momentos antes de ser gravado o registro, com é o caso do evento **BeforePost**.

- Para confirmarmos as alterações para a base SQL, pressione novamente a tecla F11 e na página *Events*, dê um duplo click sobre o evento **AfterPost**:

```
procedure TDM_Basico.TabBasicoAfterPost(DataSet: TDataSet);
begin
  if F_Menu.DBDisco.IsSQLbased then      Se a base de dados é padrão SQL
  begin
    F_Menu.DBDisco.Commit;              Gravando as alterações da tabela
    F_Menu.DBDisco.StartTransaction;    Reinicia o modo de transações
  end;
end;
```

Validando os Campos

Para não acontecer um duplicação dos nomes dos CD's, utilizaremos para a crítica a mesma idéia do que aconteceu com o formulário de Categoria:

1. Marque o objeto *TabBasicoNOM_DISCO*, e dê um duplo clique sobre o evento **OnValidate**:

```
procedure TDM_Basico.TabBasicoNom_DiscoValidate(Sender: TField);
begin
  if DSBasico.State in [dsEdit, dsInsert] then      Se o modo é de inserção ou edição
  if TabBasicoConf.FindKey([TabBasicoNOM_DISCO]) then  Pesquisa o campo digitado
  begin
    F_Basico.EditNOM_DISCO.SetFocus;                Altera a posição do cursor
    raise Exception.Create('Nome do CD duplicado'#10+  Caso já exista mostra mensagem
      'Click no botão "Localiza" em caso de dúvida');  de erro e impede o cadastro
  end;
end;
```

- Para que o nosso usuário escolha somente as categorias existentes:
 1. Marque o objeto *TabBasicoSIG_CATEG*, e dê um duplo clique sobre o evento **OnValidate**:

```
procedure TDM_Basico.TabBasicoSig_CategValidate(Sender: TField);
begin
  if DSBasico.State in [dsEdit, dsInsert] then
  if not (TabBasicoConf.FindKey([TabBasicoSIG_CATEG])) then  Observe o comando NOT
  begin
    F_Basico.EditSIG_CATEG.SetFocus;
    raise Exception.Create('Sigla da categoria não existe'#10+

```

'Click no botão "Localiza Categoria" em caso de dúvida');

end;
end;

Com o *DataModule* o nosso trabalho já está concluído, salve o objeto e confirme o código:

unit DMBasico;

interface

uses

SysUtils, Windows, Classes, Graphics, Controls, Forms, Dialogs, DB, DBTables;

type

```
TDM_Basico = class(TDataModule)
  TabMusicaCod_Disco: TFloatField;
  TabMusicaNum_Faixa: TFloatField;
  TabMusicaNom_Musica: TStringField;
  DSBasico: TDataSource;
  TabBasico: TTable;
  TabMusica: TTable;
  DSMusica: TDataSource;
  TabBasicoConf: TTable;
  TabCategor: TTable;
  TabCategorConf: TTable;
  DSCategor: TDataSource;
  TabBasicoCod_Disco: TFloatField;
  TabBasicoNom_Disco: TStringField;
  TabBasicoTip_Disco: TStringField;
  TabBasicoFot_Capa: TBlobField;
  TabBasicoSig_Categ: TStringField;
  QryContador: TQuery;
  QryContadorMAXOFCOD_DISCO: TFloatField;
procedure DM_BasicoCreate(Sender: TObject);
procedure DM_BasicoDestroy(Sender: TObject);
procedure TabBasicoNewRecord(DataSet: TDataSet);
procedure TabBasicoBeforePost(DataSet: TDataSet);
procedure TabBasicoAfterPost(DataSet: TDataSet);
procedure TabBasicoNom_DiscoValidate(Sender: TField);
procedure TabBasicoSig_CategValidate(Sender: TField);
private
  { private declarations }
public
  { public declarations }
end;
```

var

DM_Basico: TDM_Basico;

implementation

{ \$R *.DFM }

uses

fBasico, { Cadastro do Básico }

```
fMenu; { Menu Principal }

procedure TDM_Basico.DM_BasicoCreate(Sender: TObject);
begin
  if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.StartTransaction;
  TabCategor.Open;
  TabCategorConf.Open;
  TabMusica.Open;
  TabBasico.Open;
  TabBasicoConf.Open;
end;

procedure TDM_Basico.DM_BasicoDestroy(Sender: TObject);
begin
  TabBasico.Close;
  TabBasicoConf.Close;
  TabMusica.Close;
  TabCategor.Close;
  TabCategorConf.Close;
  if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.Commit;
end;

procedure TDM_Basico.TabBasicoNewRecord(DataSet: TDataSet);
begin
  F_Basico.EditNOM_DISCO.SetFocus;
end;

procedure TDM_Basico.TabBasicoBeforePost(DataSet: TDataSet);
begin
  if DSBasico.State = dsInsert then
    begin
      QryContador.Active := False;
      QryContador.Active := True;
      with QryContador.Fields[0] do
        if IsNull then
          TabBasicoCOD_DISCO.Value := 1
        else
          TabBasicoCOD_DISCO.Value := AsInteger + 1;
    end;
end;

procedure TDM_Basico.TabBasicoAfterPost(DataSet: TDataSet);
begin
  if F_Menu.DBDisco.IsSQLbased then
    begin
      F_Menu.DBDisco.Commit;
      F_Menu.DBDisco.StartTransaction;
    end;
end;

procedure TDM_Basico.TabBasicoNom_DiscoValidate(Sender: TField);
begin
  if DSBasico.State in [dsEdit, dsInsert] then
    if TabBasicoConf.FindKey([TabBasicoNOM_DISCO]) then
```

```
begin
  F_Basico.EditNOM_DISCO.SetFocus;
  raise Exception.Create('Nome do CD duplicado'#10+
    'Click no botão "Localiza" em caso de dúvida');
end;
end;

procedure TDM_Basico.TabBasicoSig_CategValidate(Sender: TField);
begin
  if DSBasico.State in [dsEdit, dsInsert] then
    if not (TabCategorConf.FindKey([TabBasicoSIG_CATEG])) then
      begin
        F_Basico.EditSIG_CATEG.SetFocus;
        raise Exception.Create('Sigla da categoria não existe'#10+
          'Click no botão "Localiza Categoria" em caso de dúvida');
      end;
    end;
  end;
end.
```

Alterando a Janela Criada

Vamos novamente alterar a janela criada, como já foi dito, vou usar um padrão de janela que achei como ideal, mas você poderá, futuramente, também encontrará o seu próprio padrão, então vamos as alterações, antes, lembre-se, é preciso entender e compreender totalmente o capítulo anterior e o *DataModule* criado deve ter passado pela verificação sem erros, para verificar um objeto a partir do menu principal escolha as opções **Project | Syntax Check**.

Deve ocorrer um erro na cláusula **Uses** informando que a unidade **fBasico** não existe nem o objeto **F_Basico** não foi encontrado, então antes de qualquer passo chame o objeto **Form2** e altere a propriedade **Name** para **F_Basico** e salve o formulário com o nome de **fBasico**, pode novamente verificar a unidade **DMBasico** que desta vez não haverá problemas.

U' Aproveite o objeto **F_Basico** para alterar a cláusula **uses** logo abaixo da diretiva de compilação para:

```
{ $R *.DFM }
```

```
uses
```

```
  DMBasico; { Referência ao DataModule }
```


Organizando os Panels

Vamos passar para as alterações com objeto *F_Basico*. Os objetos da janela se encontram distribuídos em três objetos **Panel**, o primeiro (*Panel1*) guarda o objeto *DBNavigator*, o segundo (*Panel2*) guarda os labels e os campos do arquivo **BASICO** e o terceiro (*Panel3*) guarda o objeto *DBGrid1* que controlará o arquivo **MUSICA**.

- Altere a propriedade **Align** do objeto *Panel3*, para **alNone**.
- Aumente a janela do modo que todos os campos do objeto mestre apareçam.

U' Coloque os objetos *label4* e *ImageFOT_CAPA* a direita dos outros campos.

U' Depois de tudo arrumado altere a propriedade **AutoScroll** no objeto *ScrollBar* para **True** depois para **False**, isto fará com que a barra de rolagem vertical desapareça.

Crie um quarto objeto *Panel*  para colocar os botões, conforme as instruções do capítulo anterior, para facilitar o trabalho chame o objeto *F_Categ*, através do *Project Manager*, clique sobre o objeto *Panel3* e pressione *Ctrl+C* chame novamente o objeto *Form2* e pressione *Ctrl+V*, crie um terceiro botão e altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	&Músicas	Label do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTON S\CDDRIVE.BMP	Imagem a ser mostrada no objeto
Height	25	Altura do objeto
Hint	Cadastro e manutenção das músicas	Ajuda on-line para o objeto específico
Name	ButMusica	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Width	89	Tamanho do objeto

- Reorganize os três botões no objeto *Panel4*

Modificando os campos e Labels

- Exclua o label (objeto *Label1*) e o campo (objeto *EditCOD_DISCO*) que faz referência ao COD_DISCO.
- Modifique a fonte dos campos e labels conforme descrito no capítulo anterior.
- Altere as propriedades **Caption** dos *Labels* para: “&Nome:”, “&Tipo:”, “Cate&goria:” e “&Capa:” respectivamente
- Altere a propriedade **AutoSize** dos *Labels* para **True**
- Modifique a propriedade **Stretch** do objeto *ImageFOT_CAPA* para **true**, isto fará com que a imagem da capa fique sempre de acordo com o tamanho do objeto.

Organizando os Panels

Vamos organizar cada objeto *Panel* por partes:

Objeto *Panel1*


- Modifique o objeto *DBNavigator* conforme descrito no capítulo anterior.

- Para a propriedade **Hints**, escreva novamente o auxílio para cada botão ou, chame o formulário *F_Categ* e copie as descrições da propriedade **Hints** com *Ctrl+C* e chame novamente o objeto DBNavigator e digite *Ctrl+V* dentro da propriedade.
- Modifique também o objeto *Panel1* conforme descrito no capítulo anterior e altere a propriedade **Caption** para **CD's**.

Objeto *Panel2*

- Altere as propriedades **Height** e **Width** para 216 do objeto *ImageFOT_CAPA*.
- Crie um novo objeto *Label* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	&Músicas:	Label do objeto
Font	MS Sans Serif, Negrito, 8, Castanho	Tipo de letra a ser mostrada no objeto
FocusControl	DBGrid1	Controle do foco

- Aumente o objeto *Panel2* de forma a caber os outros objetos, organize os objetos da seguinte forma: Nome, Tipo, Categoria e Músicas, ao lado coloque a foto. Deixe o espaço de um campo entre Categoria e Músicas.
- Remova o objeto *EditTIP_DISCO* e em seu lugar crie o objeto *DBComboBox* , encontrado na *Component Palette* página *Data Controls*, e altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSource	DM_Basico.DSBasico	DataSource vinculado
DataField	TIP_DISCO	Campo de tabela
Font	MS Sans Serif, Normal, 8, Azul Marinho	Tipo de letra a ser mostrada no objeto
Hint	Selecione o tipo	Ajuda on-line para o objeto específico
Items	AAA; AAD; ADD; e DDD	Itens que aparecerão como opções do COMBO BOX, coloque um em cada linha.
Name	ComboTIP_DISCO	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line


- No objeto *Label3* recoloque a propriedade **FocusControl** apontando para o objeto *ComboTIP_DISCO*.

Objeto *Panel3*

- Marque o objeto DBGrid1 e altere as propriedades **Align** para *alNone* e **BorderStyle** para *bsSingle*, pressione *Ctrl+X* marque o objeto **ScrollBar** e pressione *Ctrl+V*, acerte o objeto de forma que este caiba abaixo do objeto *Label1* (Músicas:).
- Elimine o objeto *Panel3*, clique sobre ele e pressione *Del*.

Modificando a Janela

- Altere as propriedades **BorderIcons**, **BorderStyle** e **Position** conforme descrito no capítulo anterior.

- Mude a propriedade **Caption** para **Cadastro**.
- Altere a propriedade **Align** do objeto **Panel2** para **alClient** e acerte as posições no formulário.
- Salve o formulário e salve o projeto.
- No espaço deixado entre os labels de Categoria e Música, crie o objeto **DBText**  encontrado na *Component Palette* página *Data Controls*, que servirá para mostrar o nome da categoria selecionada, altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSource	DM_Basico.DSCategor	DataSource vinculado
DataField	DES_CATEG	Campo de tabela
Font	MS Sans Serif, Normal, 8, Castanho	Tipo de letra a ser mostrada no objeto

- Crie agora três objetos **SpeedButton** , encontrado na *Component Palette* página *Additional*:

Para o objeto SpeedButton1: Posicione-o ao abaixo do label Capa.

Propriedade	Valor	Descrição
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\ANIMATN.BMP	Imagem a ser mostrada no objeto
Height	25	Altura do objeto
Hint	Cópia imagem da área de transferência	Ajuda on-line para o objeto específico
Name	ButPaste	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Width	25	Tamanho do objeto

Para o objeto SpeedButton2: Posicione-o ao abaixo do objeto *EditCOD_CATEG*

Propriedade	Valor	Descrição
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\LANTERNA.BMP (veja dica)	Imagem a ser mostrada no objeto
Height	25	Altura do objeto
Hint	Pesquisa determinada categoria	Ajuda on-line para o objeto específico
Name	ButLocCateg	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Width	25	Tamanho do objeto


Para o objeto SpeedButton3: Posicione-o ao lado do objeto *ButLocCateg*

Propriedade	Valor	Descrição
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\CRDFILE2.BMP	Imagem a ser mostrada no objeto
Height	25	Altura do objeto
Hint	Inserir registro na tabela de	Ajuda on-line para o objeto específico

	categoria	
Name	ButInsCateg	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Width	25	Tamanho do objeto




U' A função de cada botão está explicada na propriedade *hint*, para os objetos *ButLocCateg* e *ButInsCateg* iremos nos aproveitar dos formulários já construídos anteriormente.

U' Provavelmente você não irá encontrar o arquivo LANTERNA.BMP solicitado para o segundo botão, teremos de criá-lo:

1. Salve o formulário e feche o projeto, localize (no diretório demos do *Delphi*) e abra o projeto [DiretórioDelphi]\DEMOS\DB\MASTAPP\MASTAPP.DPR
2. Através do *Project Manager* abra o formulário *SearchDlg*.
3. Localize o objeto *SearchButton* e clique  na propriedade **Glyph**.
4. Escolha o botão **Save...** e salve-o no diretório e nome proposto ([DiretórioDelphi]\IMAGES\BUTTONS\ e LANTERNA.BMP).
5. Retorne ao nosso projeto descartando quaisquer aviso para salvar o MASTAPP.DPR.

Trabalhando com Grid's

Falemos agora de um objeto especial o Grid, especial por ser um dos objetos de todo o conjunto do *Delphi* o mais prático e fácil de usar, quem não se lembra no velho *Clipper* da função **dbEdit()**, quem não utilizou seus recursos para mostrar registros ou realizar consultas com filtros especiais. O objeto *Grid* é o “neto” deste objeto com alguns recursos mais simplificados, no total são três os objetos *Grid*'s:

1. *StringGrid*  - Componente da unidade *Grid* sendo utilizado para, de forma simplificada, associar *Strings* a uma grade contendo linhas e colunas, encontrado na *Component Palette* página *Additional*.
2. *DrawGrid*  - Componente da unidade *Grid* que permite mostrar uma estrutura de dados existentes no formato de linhas e colunas, encontrado na *Component Palette* página *Additional*.
3. *DBGrid*  - Componente da unidade *DBGrids* que mostrar dados de um *DataSet* para um componente no formato de linhas e colunas, encontrado na *Component Palette* página *Data Controls*.

Por enquanto vou me deter a falar do objeto *DBGrid*, mais tarde voltaremos a falar dos outros, este objeto *DBGrid* é vinculado, ao *DataSource*, apresenta as seguintes propriedades (mais importantes, algumas ainda não mencionadas):

- ✓ *DataSource*: Nome do objeto *DataSource* vinculado;
- ✓ *Font*: Tipo da letra a ser mostrada no conteúdo do objeto;

- ✓ Options: série de opções de controle (se a opção *True* for selecionada):
 - ♦ **dgEditing**: permite a edição e adição dos dados;
 - ♦ **dgAlwaysShowEditor**: O grid entra automaticamente em modo de edição, não havendo a necessidade de pressionar **Enter** ou **F2** (depende que a propriedade **dgEditing = True**);
 - ♦ **dgTitles**: Viabiliza o uso do título de cada campo;
 - ♦ **dgIndicator**: Habilita o ponteiro de indicação da coluna;
 - ♦ **dgColumnResize**: A coluna pode ser redimensionada;
 - ♦ **dgColLines**: Habilita a separação das colunas;
 - ♦ **dgRowLines**: Habilita a separação das linhas;
 - ♦ **dgTabs**: Use o pressionamento das teclas *Tab* e *Shif+Tab* para se mover dentro das colunas;
 - ♦ **dgRowSelect**: Seleciona, com uma tarja azul, todas as colunas de uma linha;
 - ♦ **dgAlwawsShowSelection**: As células do *grid* são mostradas constantemente selecionadas, mesmo que este não detenha o foco.
 - ♦ **dgConfirmDelete**: Use as teclas *Ctrl+Del*, para excluir dados;
 - ♦ **dgCancelOnExit**: Se qualquer inclusão estiver pendente e for dado saída no grid sem a validação dos dados, a inclusão é cancelada. Previne a inclusão de registros inválidos ou em branco.
- ✓ **TitleFont**: Tipo da letra a ser mostrada nos títulos do objeto.

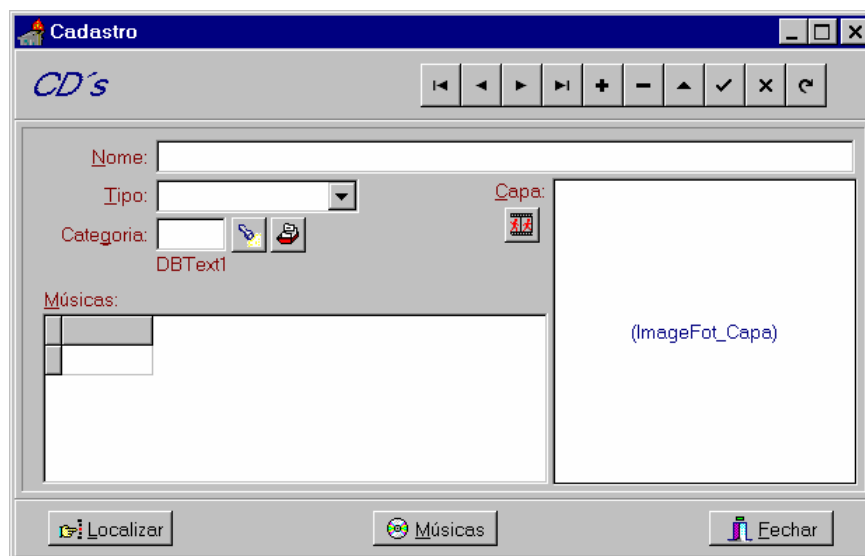
U' Os campos no objeto *DBGrid* são adicionados e controlados através do *FieldsEditor*

O nosso objeto *grid* mostrará apenas as músicas de cada CD, para tanto altere as seguintes propriedades:

Propriedade	Valor	Descrição
Font	MS Sans Serif, Negrito, 8, Castanho	Tipo de letra a ser mostrada no objeto
Hint	Músicas deste CD	Ajuda on-line para o objeto específico
Options	[dgTitles, dgIndicator, dgColLines, dgRowLines, dgRowSelect, dgAlwawsShowSelection]	Opções de controle
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line

U' Não lhe aconselho usar este objeto para realizar alterações em bases de dados (apesar de isto ser possível) é mais prático utilizar uma janela separada para realizar este trabalho, as idéias ficarão mais bem ordenadas, se cada formulário controlar uma única entrada em tabela de cada vez.


- Se você está meio perdido com isto tudo, não se desespere, simplesmente compare os formulários para ver se não falta nada:



Finalmente, a programação

Agora falta apenas o código, pelo tamanho do formulário e pelo número de controles já dá para pensarmos que precisamos programar linhas e linhas de código, engano ! o maior trabalho já foi feito, observe:

- Criando o *DataModule* ao ser chamado formulário:

1. Clique no botão  (Toggle Form/Unit) da *SpeedBar*, até você alternar para o *Code Editor*.

```

public
procedure inicio;                                Criando a chamada para um procedimento público.
end;

var
  F_Basico: TF_Basico;

implementation

{$R *.DFM}

procedure TF_Basico.Inicio;                        Crie o início do procedimento
begin
  DM_Basico := TDM_Basico.Create(Application);     Cria o DataModule
  Screen.Cursor := crDefault;                     Faz o cursor ficar no formato de Seta
  ShowModal;                                       Mostra o formulário F_Basico
end;

```

- Código para eliminar o *DataModule*, basicamente é o mesmo trabalho realizado com o objeto **F_Basico**:

1. No objeto *F_Basico*, dê um duplo click sobre o evento **OnClose**:

```

procedure TF_Basico.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if F_Menu.CriticaEdicao(DM_Basico.DSBasico, 'CD', 'Fechar') then      Lembra da Função
  begin
    Action := caNone;          Cancela a saída da janela
    Exit;                      Sai da procedure
  end;
  Screen.Cursor := crHourGlass;    Faz o cursor virar uma ampulheta
  DM_Basico.Free;                 Elimina o DataModule
end;

```

U' Lembre-se que a função **CriticaEdicao** faz parte da Unit **fMenu**, então é preciso declarar esta unidade com o comando **USES** (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

```

  DMBasico, { Referência ao DataModule }
  fMenu;    { Menu Principal do Sistema }      Colocado para o uso da Função CriticaEdicao

```

U' Repare que no início desta **Unit** existe uma declaração **Uses**, após o comando **interface**. Então por que não colocar todas essas declarações em um lugar só? 1.O *Delphi* controlará (colocando ou removendo) estas Units ali colocadas (dependendo dos objetos utilizados) e 2.Todos os comandos declarados antes da declaração **implementation** (com exceção de eventos de criação. Ex.: **onCreate**), serão executados e **objetos** e **units** ficarão em memória esperando serem chamados, então é impraticável colocar **units** que só serão utilizadas em tempo de execução.

Consulta

- Código para pesquisar os registros das tabelas, basicamente utilizaremos o mesmo trabalho realizado com o objeto **F_Categ**:

1. Dê um duplo click sob o objeto *ButLocalizar*:

```

procedure TF_Basico.ButLocalizarClick(Sender: TObject);
var
  ObjPesquisa: String;
begin
  if F_Menu.CriticaEdicao(DM_Basico.DSBasico, 'CD', 'Localizar') then      Lembra da Função
  begin
    Exit;

    ObjPesquisa := TabBasicoNOM_DISCO.Value;

    if InputQuery('Pesquisa','Entre com o nome do CD (ou parte).', ObjPesquisa) then
      TabBasico.FindNearest([ObjPesquisa]);
  end;

```

U' Lembre-se que a função **InputQuery** faz parte da Unit **Dialogs**, então é preciso declarar esta unidade com o comando **USES** (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

DMBasico, { Referência ao DataModule }	
fMenu, { Menu Principal do Sistema }	
Dialogs; { Gerente de Mensagens }	Colocado para o uso da Função InputQuery

U' Desta vez foi utilizado o comando **FindNearest**, este comando consulta por parte inicial do código encontrado um código igual ou maior que o pesquisado, não é preciso ao usuário lembrar o nome completo como no comando **FindKey**. **Atenção:** o comando **FindNearest** não retornará uma variável **boolean** (True ou False como resultado da pesquisa) então não se faz a necessidade de críticas sobre o mesmo. Salvo exceção se o campo for tipo numérico ou tipo data então utilize este comando em conjunto com o comando **Try**. Veja no próximo capítulo como.

- O código para o botão que localizará um registro na tabela de categoria será tratado no próximo capítulo.
- Para inserir novos registros na tabela de categoria utilizaremos o formulário construído anteriormente, a única diferença é que desta vez devemos estar em modo de edição:
 1. Dê um duplo click sobre o objeto *ButInsCateg*:

procedure TF_Basico.ButInsCategClick(Sender: TObject);

begin

if not (DSBasico.State in [dsEdit, dsInsert]) then	Se não estiver em estado de edição ou inclusão
begin	
MessageDlg('Você não está no modo de edição!',	Envia mensagem de erro
mtInformation, [mbOK], 0);	

 exit;

end

Screen.Cursor := crHourGlass;	Faz o cursor virar uma ampulheta
-------------------------------	----------------------------------

if F_Menu.DBDisco.IsSQLbased **then**

 F_Menu.DBDisco.Commit;

Gravamos as alterações no banco de dados

with TF_Categ.Create(Self) **do**

Cria o formulário na área de memória

begin

 Inicio;

Chama o formulário

 Free;

Elimina o formulário da memória

end;

if F_Menu.DBDisco.IsSQLbased **then**

Se a base de dados é padrão SQL

 F_Menu.DBDisco.StartTransaction;

Inicia o modo de transações

Screen.Cursor := crDefault;

Faz o cursor ficar no formato de Seta

end;

U' Não esqueça de declarar a **Unit** deste formulário:

{ \$R *.DFM }

uses

DMBasico, { Referência ao DataModule }
fMenu, { Menu Principal do Sistema }
Dialogs, { Gerente de Mensagens }
fCateg; { Cadastro de Categorias }

Trabalhando com a área de Transferência


Na maioria dos aplicativos para Windows é comum encontrarmos os comandos Recortar, Copiar e Colar, o famoso **Cut/Copy/Paste**, isto é realizado utilizando a área de transferência do Windows:

São três os comandos que fazem o trabalho na área de transferência:

- ✓ Objeto.**CutToClipboard**, envia o conteúdo do objeto para área de transferência; e
 - ✓ Objeto.**CopyToClipboard**, faz uma cópia do conteúdo do objeto na área de transferência.
 - ✓ Objeto.**PasteFromClipboard**, traz para o conteúdo do objeto da área de transferência;
- Próximo passo, o código para o botão que permitirá trazer uma imagem armazenada na área de transferência do Windows (Salva com o utilitário PaintBrush, por exemplo) para o campo da Foto:
 1. Dê um duplo click sobre o objeto *ButPaste*:

```
procedure TF_Basico.ButPasteClick(Sender: TObject);
begin
  ImageFOT_CAPA.PasteFromClipboard;
end;
```

Utilizando o objeto OpenFileDialog

Um aluno uma vez me sugeriu uma outra maneira de colocar a figura, ao invés de se utilizar da área de transferência, poderíamos chamar o arquivo (.BMP) diretamente, isto é realizado com o uso de um objeto *OpenDialog* , encontrado na *Component Palette* página *Dialog*, coloque-o no formulário e altere as seguintes propriedades:

Propriedade	Valor	Descrição
Filter	Arquivos bitmap *.BMP	Seleciona os tipos de arquivo que serão abertos
Name	AbreBmp	Nome do Objeto

- Próximo passo, o código para o botão que permitirá trazer uma imagem em arquivo .BMP para o campo da Foto:
 1. Dê um duplo click sobre o objeto *ButPaste*:

```
procedure TF_Basico.ButPasteClick(Sender: TObject);
begin
  If AbreBmp.Execute then                               Verifica se foi clicado em OK
    ImageFot_Amostra.Picture.LoadFromFile( AbreBmp.FileName ); Carrega o arquivo
end;
```

U'Escolha a maneira que mais lhe agrada.

U'Outro objeto interessante é o objeto **SaveDialog**, veja outras utilizações sobre eles no **apêndice F**

Criando o formulário para o cadastro das músicas

Construiremos o formulário para cadastrar as músicas, crie o formulário com base no formulário criado para a tabela de categoria e faça as seguintes alterações:

- ✓ Altere a propriedade **name** do formulário para **F_MUSICA**.
- ✓ Chame o *DataModule* **DMBasico** e crie mais um objeto *TTable*, para fazermos as críticas das músicas duplicadas e altere as seguintes propriedades:

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Banco de Dados ou a localização do diretório das tabelas
TableName	MUSICA	Nome externo da tabela
Name	TabMusicaConf	Nome do objeto
IndexFieldNames	Cod_Disco;Num_Faixa	Nome do campo indexado
ReadOnly	True	Somente leitura

U'Não esqueça de alterar o programa do *DataModule* para abrir e fechar a nova tabela e coloque a crítica para a tabela de música, evento **OnValidate** do objeto **TabMusicaNum_Faixa**:

```

procedure TDM_Basico.TabMusicaNum_FaixaValidate(Sender: TField);
begin
  if DSMusica.State in [dsEdit, dsInsert] then
    begin
      TabMusicaConf.EditKey;
      TabMusicaConf.FieldByName('COD_DISCO').AsFloat := TabMusicaCod_Disco.Value;
      TabMusicaConf.FieldByName('NUM_FAIXA').AsFloat := TabMusicaNum_Faixa.Value;
      if TabMusicaConf.GotoKey then
        begin
          F_Musica.EditNum_Faixa.SetFocus;
          raise Exception.Create('Faixa do CD duplicada');
        end;
      end;
    end;
end;

```

U'Repare no uso do comando **GotoKey** ao invés do comando **FindKey**, ele foi utilizado por se tratar de uma chave composta.

U'Coloque também a crítica para a tabela de música, evento **OnNewRecord** do objeto **TabMusica**:

```

procedure TDM_Basico.TabMusicaNewRecord(DataSet: TDataSet);
begin
  F_Musica.EditNUM_FAIXA.SetFocus;
end;

```

U'Uma última crítica será para confirmarmos as alterações para a base SQL, pressione novamente a tecla F11 e na página *Events*, dê um duplo click sobre o evento **AfterPost**:

```

procedure TDM_Basico.TabMusicaAfterPost(DataSet: TDataSet);
begin
  if F_Menu.DBDisco.IsSQLbased then           Se a base de dados é padrão SQL
    begin

```

```

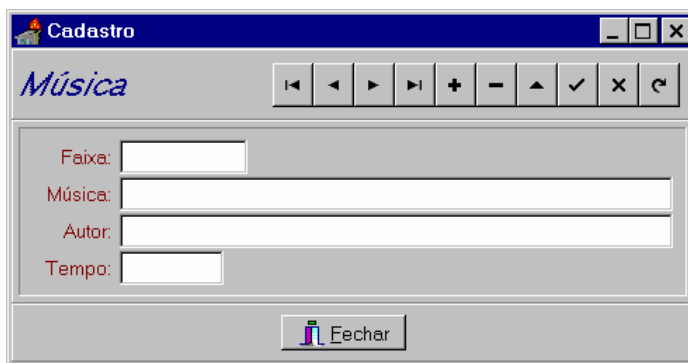
F_Menu.DBDisco.Commit;
F_Menu.DBDisco.StartTransaction;
end;
end;

```

Gravando as alterações da tabela

Reinicia o modo de transações

- ✓ Deixe apenas o botão Fechar;
 - ✓ Remova o Label e o DBEdit do campo COD_DISCO; e
 - ✓ Mude a propriedade do *BorderStyle* para **bsDialog**.
- Observe e compare com o desenho do formulário abaixo:



- Altere inicialmente, a cláusula *uses* para o **DMBasico**, em seguida a propriedade de todos os campos e do DBNavigator para **DM_Basico.DSMusica**.
- A programação do formulário é basicamente a mesma para as tabelas, então ao invés de perder tempo e espaço falando tudo de novo, observe o código completo e note que a única mudança está em prevenir para que o formulário não seja fechado em tempo de edição ou inserção e com o estado do cursor:

```
unit fMusica;
```

```
interface
```

```
uses
```

```

SysUtils, Windows, Messages, Classes, Graphics, Controls,
StdCtrls, Forms, DBCtrls, DB, Buttons, Mask, ExtCtrls;

```

```
type
```

```

TF_Musica = class(TForm)
  ScrollBox: TScrollBox;
  Label2: TLabel;
  EditNum_Faixa: TDBEdit;
  Label3: TLabel;
  EditNom_Musica: TDBEdit;
  Label4: TLabel;
  EditNom_Autor: TDBEdit;
  Label5: TLabel;
  EditTmp_Musica: TDBEdit;
  DBNavigator: TDBNavigator;
  Panel1: TPanel;
  Panel2: TPanel;
  Panel3: TPanel;

```

```

    ButFechar: TBitBtn;
procedure FormShow(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
    { private declarations }
public
    { public declarations }
end;

var
    F_Musica: TF_Musica;

implementation

{$R *.DFM}

uses
    DMBasico; { Referência ao DataModule }

procedure TF_Musica.FormShow(Sender: TObject);
begin                                     Evento OnShow do objeto F_Musica
    Screen.Cursor := crDefault;
end;

procedure TF_Musica.FormClose(Sender: TObject; var Action: TCloseAction);
begin                                     Evento OnClose do objeto F_Musica
    if F_Menu.CriticaEdicao(DM_Basico.DSMusica, 'Música', 'fechar' ) then
        begin
            Action := caNone;
            exit;
        end;
    Screen.Cursor := crHourGlass;
end;

end.

```

U' Não é preciso se preocupar com a gravação para o campo COD_DISCO, isto será feito automaticamente pelo *Delphi*, mantendo a integridade referencial.

- Salve o formulário com o nome *fMusica*.

U' Vamos agora alterar o botão do formulário principal, objeto *butMusica*, que será usado para chamar este segundo.

```

procedure TF_Basico.ButMusicaClick(Sender: TObject);
begin
    if DM_Basico.DSBasico.State in [dsEdit, dsInsert] then
        begin
            MessageDlg('Salve a edição do CD antes de Editar as músicas!', mtInformation, [mbOk], 0);
            Exit;
        end;
    Screen.Cursor := crHourGlass;
    if F_Menu.DBDisco.IsSQLbased then
        F_Menu.DBDisco.Commit;
    with TF_Musica.Create(Self) do           Criamos o formulário em memória
        begin

```

```

    ShowModal
    Free;
end;
if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.StartTransaction;
Screen.Cursor := crDefault;
end;

```

Chamaremos aqui o novo formulário
Eliminamos o Formulário

U' Não esqueça de adicionar a propriedade **uses**
{ \$R *.DFM }

```

uses
    DMBasico,      { Referência ao DataModule }
    fMenu,         { Menu Principal do Sistema }
    Dialogs,       { Gerente de Mensagens }
    fCateg,        { Cadastro de Categorias }
    fMusica;       { Cadastro de Músicas }

```

U' Não esqueça também de retirar os objetos *F_Musica* e *F_Basico* da criação automática, opção do menu principal clique em **Options** | **Project...**, qualquer dúvida consulte o capítulo anterior.

Criando novos Procedimentos Globais

Vamos criar mais alguns procedimentos globais, para a nosso primeiro procedimento note as chamadas aos formulários externos *F_Categ* e *F_Musica*:

```

Screen.Cursor := crHourGlass;
if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.Commit;
...
...
if F_Menu.DBDisco.IsSQLbased then
    F_Menu.DBDisco.StartTransaction;
Screen.Cursor := crDefault;

```

Os comandos listados se repetem tanto para o procedimento **ButInsCategClick** e para o procedimento **ButMusicaClick**, chame o formulário **F_Menu** e crie o seguinte procedimento:

```

public
    procedure Prepara(Tipo: boolean);
    function CriticaEdicao(DSOrigem: TDataSource; DoQue, AntesDe: String) : boolean;
end;

```

Aqui

```

var
...
...
procedure TF_Menu.Prepara(Tipo: boolean);
begin
    if tipo then
        begin
            Screen.Cursor := crHourGlass;
            if DBDisco.IsSQLbased then
                DBDisco.Commit;

```

Se o valor da variável **tipo** recebida for verdadeiro
Faz a primeira parte

```

end
else
begin
    if DBDisco.IsSQLbased then
        DBDisco.StartTransaction;
        Screen.Cursor := crDefault;
    end;
end;

```

Modifique agora ambos os procedimentos do objeto **F_Basico**:

```

procedure TF_Basico.ButInsCategClick(Sender: TObject);
begin
    ...
    ...
    F_Menu.Prepara(True);
    with TF_Categ.Create(Self) do
    begin
        Inicio;
        Free;
    end;
    F_Menu.Prepara(False);
end;

```

```

procedure TF_Basico.ButMusicaClick(Sender: TObject);
begin
    ...
    ...
    F_Menu.Prepara(True);
    with TF_Musica.Create(Self) do
    begin
        Inicio;
        Free;
    end;
    TabMusica.Refresh;
    F_Menu.Prepara(False);
end;

```

Um segundo procedimento global pode ser criado verificando os eventos **AfterPost** dos DataModules *Dm_Categ* e *DM_Basico* (este segundo em dois lugares); observamos o mesmo procedimento:

```

if F_Menu.DBDisco.IsSQLbased then
begin
    F_Menu.DBDisco.Commit;
    F_Menu.DBDisco.StartTransaction;
end;

```

Criamos então o seguinte procedimento no formulário *F_Menu*:

```

public
    procedure GravaBanco
    procedure Prepara(Tipo: boolean);
    function CriticaEdicao(DSOrigem: TDataSource; DoQue, AntesDe: String) : boolean;
end;

```

```

var
    ...

```

```

...
procedure TF_Menu.GravaBanco;
begin
  if DBDisco.IsSQLbased then      Embutiremos os comandos puros aqui
  begin
    DBDisco.Commit;
    DBDisco.StartTransaction;
  end;
end;

```

Agora substitua os eventos nos dos DataModules *Dm_Categ* e *DM_Basico*; pela seguinte chamada ao procedimento

```

begin
  F_Menu.GravaBanco;
end;

```

U' Repare também que foi retirado das funções que ficaram no formulário *F_Menu* as referências da base de dados ao formulário (eram **F_Menu.DBDisco** e ficou simplesmente **DBDisco**).

Alterando o Menu para receber o formulário

Agora finalmente vamos rodar nosso formulário, para tanto precisamos chamá-lo através do menu principal para isto:

1. Chame novamente o formulário **F_Menu**: no menu principal escolha **V**iew e **P**roject **M**anager, marque o objeto e clique sobre o botão **V**iew **f**orm.
2. Clique em **A**rquivo, **C**adastro, coloque o seguinte código:

```

procedure TF_Menu.Cadastro1Click(Sender: TObject);
begin
  Screen.Cursor := crHourGlass;      Transforma o cursor em uma ampulheta
  F_Basico := TF_Basico.Create(Application);  Cria o formulário em memória
  F_Basico.Inicio;                   Chama o formulário através da área aberta
  F_Basico.Free;                     Libera a área aberta
  Screen.Cursor := crDefault;        Transforma o cursor em uma seta
end;

```

3. O objeto *F_Basico* faz parte da Unit **fBasico** então é necessário fazer o uso desta Unit, para tanto insira o seguinte código (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

```

fSobre, { Janela do Sobre o Sistema ... }
fCateg, { Utilizada para o cadastro de Categorias }
fBasico; { Utilizada para o cadastro de CD's }

```

U' Não esqueça de retirar o formulário *F_Basico* da lista de formulários criados automaticamente.

4. Saia do *Code Editor* e salve o formulário e o projeto.

5. Podemos agora salvar e testar o formulário completo, para incluir algumas imagens utilize o *PaintBrush* do Windows, ou outro utilitário qualquer.
6. Se alguma coisa deu errada, releia o capítulo, ou então confira todo o código:

```
unit fBasico;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics, Controls,  
StdCtrls, Forms, DBCtrls, DB, DBGrids, Buttons, Grids, Mask, ExtCtrls;
```

```
type
```

```
TF_Basico = class(TForm)  
  ScrollBox: TScrollBox;  
  Label2: TLabel;  
  EditNom_Disco: TDBEdit;  
  Label3: TLabel;  
  Label4: TLabel;  
  ImageFot_Capa: TDBImage;  
  Label5: TLabel;  
  EditSig_Categ: TDBEdit;  
  DBNavigator: TDBNavigator;  
  Panel1: TPanel;  
  Panel2: TPanel;  
  Panel4: TPanel;  
  ButFechar: TBitBtn;  
  ButLocalizar: TBitBtn;  
  ButMusica: TBitBtn;  
  Label1: TLabel;  
  ComboTIP_DISCO: TDBComboBox;  
  DBGrid1: TDBGrid;  
  DBText1: TDBText;  
  ButLocCateg: TSpeedButton;  
  ButInsCateg: TSpeedButton;  
  ButPaste: TSpeedButton;  
  { AbreBmp: TOpenDialog; ← Se você utilizou o objeto OpenFileDialog }  
  procedure FormClose(Sender: TObject; var Action: TCloseAction);  
  procedure ButLocalizarClick(Sender: TObject);  
  procedure ButInsCategClick(Sender: TObject);  
  procedure ButPasteClick(Sender: TObject);  
  procedure ButMusicaClick(Sender: TObject);  
private  
  { private declarations }  
public  
  procedure inicio;  
end;
```

```
var
```

```
F_Basico: TF_Basico;
```

```
implementation
```

```
{ $R *.DFM }
```

```
uses
```

```
DMBasico,    { Referência ao DataModule }  
fMenu,       { Menu Principal do Sistema }  
Dialogs,     { Gerente de Mensagens }  
fCateg,      { Cadastro de Categorias }  
fMusica;     { Cadastro de Músicas }
```

```
procedure TF_Basico.Inicio;
```

```
begin
```

```
    DM_Basico := TDM_Basico.Create(Application);
```

```
    Screen.Cursor := crDefault;
```

```
    ShowModal;
```

```
end;
```

```
procedure TF_Basico.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
```

```
    if F_Menu.CriticaEdicao(DM_Basico.DSBasico, 'CD', 'Fechar') then
```

```
        begin
```

```
            Action := caNone;
```

```
            Exit;
```

```
        end;
```

```
    Screen.Cursor := crHourGlass;
```

```
    DM_Basico.Free;
```

```
end;
```

```
procedure TF_Basico.ButLocalizarClick(Sender: TObject);
```

```
var
```

```
    ObjPesquisa: String;
```

```
begin
```

```
    if F_Menu.CriticaEdicao(DM_Basico.DSBasico, 'CD', 'Localizar') then
```

```
        Exit;
```

```
    ObjPesquisa := DM_Basico.TabBasicoNom_Disco.Value;
```

```
    if InputQuery('Pesquisa','Entre com o nome do CD (ou parte).', ObjPesquisa) then
```

```
        DM_Basico.TabBasico.FindNearest([ObjPesquisa]);
```

```
end;
```

```
procedure TF_Basico.ButInsCategClick(Sender: TObject);
```

```
begin
```

```
    if not (DM_Basico.DSBasico.State in [dsEdit, dsInsert]) then
```

```
        begin
```

```
            MessageDlg('Você não está no modo de edição!',mtInformation,[mbOK],0);
```

```
            exit;
```

```
        end;
```

```
    F_Menu.Prepara(True);
```

```
    with TF_Categ.Create(Self) do
```

```
        begin
```

```
            Inicio;
```

```
            Free;
```

```
        end;
```

```
    F_Menu.Prepara(False);
```

```
end;
```

```
procedure TF_Basico.ButPasteClick(Sender: TObject);
```

```
begin
```



```
ImageFOT_CAPA.PasteFromClipboard;  
end;
```

{ Ou se você utilizou o objeto OpenFileDialog

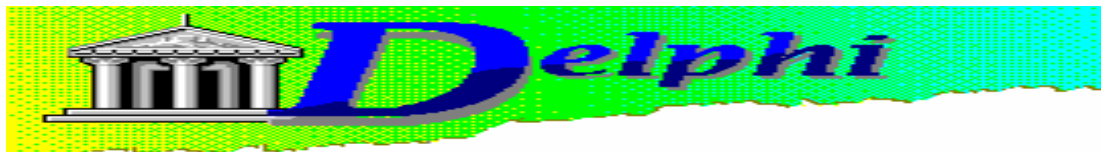
```
procedure TF_Basico.ButPasteClick(Sender: TObject);  
begin  
  if AbreBmp.Execute then  
    ImageFot_Amostra.Picture.LoadFromFile( AbreBmp.FileName );  
end;
```

}

```
procedure TF_Basico.ButMusicaClick(Sender: TObject);  
begin  
  if DM_Basico.DSBasico.State in [dsEdit, dsInsert] then  
    begin  
      MessageDlg('Salve a edição do CD antes de Editar as músicas!', mtInformation, [mbOk], 0);  
      Exit;  
    end;  
  
  F_Menu.Prepara(True);  
  with TF_Musica.Create(Self) do  
    begin  
      Inicio;  
      Free;  
    end;  
  F_Menu.Prepara(False);  
end;  
  
end.
```

Comandos e suas funções, por ordem de aparição:

with [registro ou objeto] **do** [comandos] - Cria um método de referência para o **registro** ou **objeto**.
[tabela].**FindNearest** - realiza uma pesquisa aproximada na tabela.



Capítulo VII

Trabalhando com consultas


Todo um projeto pode ir por ralo abaixo caso o usuário não consiga uma maneira eficaz e eficiente para localizar seus registros perdidos, as consultas as tabelas devem auxiliar o usuário na tarefa de lembrá-lo qual o código correto para determinada categoria, e as consultas aos CD's devem ser rápidas e práticas, senão, ficaria muito mais prático e fácil pesquisar manualmente cada CD tentando encontrar determinada música do que acessar o computador para realizar tal tarefa.

Criando consultas para tabelas

O nosso botão de **Localizar** (objeto **butLocalizar**) no formulário **F_Categ**, não é nem um pouco amigável com o usuário, ele solicita apenas um código para localizar uma determinada categoria, mas se o nosso usuário não lembrar deste código? Maneira idêntica acontece com o nosso botão de pesquisa no objeto **F_Basico**.

Trabalhando com Grid's

O objeto **DBGrid** mostrado no capítulo anterior servirá perfeitamente para criarmos uma janela que permita ao usuário uma localização mais rápida e prática de um determinado registro, vamos ao trabalho:

- Inicialmente Clique no botão  (New Form) na *Speed Bar*, ou no menu principal a Clique no menu principal a opção **File** e **New...**, em *New Items*, na página *New* e clique no objeto entitulado *Form* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsDialog	Estilo da borda do formulário
Caption	Pesquisa Categoria	Label do objeto (Tarja azul do formulário)
Name	F_SelCate	Nome do objeto
Position	poScreenCenter	Posição da janela (centralizado)

- Crie para esta nova janela uma relação com o *DataModule* **DM_Modelo** colocando-o na cláusula **Uses**, abaixo da diretiva de compilação:

```
{ $R *.DFM }
```

uses

```
DMModelo; { Referência ao DataModule }
```

- Crie para esta nova janela os seguintes objetos:

DBNavigator (encontrado na página *Data Controls*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSource	DM_Modelo.DSCategor	DataSource vinculado (caso esta opção não esteja disponível abra o <i>DataModule DM_Modelo</i>)
Hints	Um em cada linha: Primeiro, Anterior, Próximo e Último	Tópicos para ajuda on-line do objeto específico
Left	8	Posição a esquerda
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
VisibleButtons	[nbFirst, nbPrior, nbNext, nbLast]	Botões visíveis
Top	8	Distância do topo
Width	113	Largura do objeto

DBGrid (encontrado na página *Data Controls*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSource	DM_Modelo.DSCategor	DataSource vinculado
Font	MS Sans Serif, normal, 8, azul marinho	Tipo da letra a ser mostrada no conteúdo do dbGrid
Left	8	Posição à esquerda
Height	145	Altura do objeto
Options	[dgTitles, dgIndicator, dgColLines, dgRowLines, dgRowSelect, dgAlwaysShowSelection]	Opções de execução para o objeto
TitleFont	MS Sans Serif, negrito, 8, Castanho	Tipo de letra a ser mostrada no título do dbGrid
Top	48	Distância do topo
Width	375	Largura do objeto

MaskEdit (encontrado na página *Additional*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
Font	MS Sans Serif, normal, 8, azul marinho	Tipo de letra a ser mostrada no objeto
Left	56	Alinhamento a esquerda
Name	EdtTrecho	Nome do objeto
Text		Conteúdo do objeto
Width	97	Largura do objeto
Top	199	Distância do topo
EditMask	>aa;0;_	Cria uma máscara de edição para o campo.

Label (encontrado na página *Standard*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	Sigla:	Label do objeto
Focus	EdtTrecho	Controle para a posição do cursor
Font	MS Sans Serif, negrito, 8, Castanho	Tipo de letra a ser mostrada no objeto
Left	8	Alinhamento a esquerda
Top	201	Distância do topo

BitBtn (encontrado na página *Additional*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
Kind	bkOK	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption, Glyph e ModalResult
Caption	&OK	Label do objeto
Font	MS Sans Serif, normal, 8, preto	Tipo de letra a ser mostrada no objeto
Height	25	Altura do objeto
Hint	Confirma a pesquisa	Ajuda on-line para o objeto específico
Left	16	Alinhamento a esquerda
Name	ButOk	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Spacing	30	Espaço entre o Glyph e o Label
Top	248	Distância do topo
Width	89	Largura do objeto

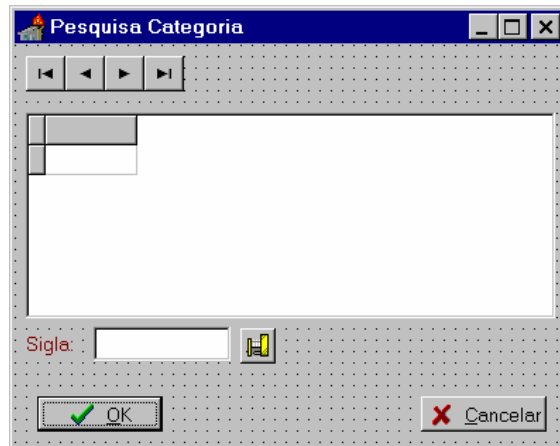
BitBtn (encontrado na página *Additional*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
Kind	bkCancel	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption, Glyph e ModalResult
Caption	&Cancela	Label do objeto
Font, Height, ShowHint, Top e Width	idêntica ao objeto ButOk	Tipo de letra a ser mostrada no objeto, altura do objeto, Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line, distância do topo e tamanho do objeto
Hint	Abandona a pesquisa	Ajuda on-line para o objeto específico
Left	288	Alinhamento à esquerda
Name	ButCancela	Nome do objeto
Spacing	8	Espaço entre o Glyph e o Label
Width	89	Largura do objeto

SpeedButton (encontrado na página *Additional*), altere as seguintes propriedades:

Propriedade	Valor	Descrição
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\FDRAWER1.BMP	Imagem a ser mostrada no objeto
Hint	Pesquisa o código selecionado na tabela	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Name	ButPesquisa	Nome do objeto
Left e Top	Modo que o objeto fique ao lado do objeto EdtTrecho	Posição esquerda e distância do topo

- Salve o formulário com o nome **fSelCate** e compare-o com o desenho abaixo:



Programando no formulário

O código para este objeto não chega a ser um troço estranho, medonho e esquisito, na verdade é até bem simples, basta apenas compreendermos o seu funcionamento, o objeto **DBGrid** fará todo o trabalho, mostrando ao usuário os registros cadastrados em uma tabela, a parte que nos resta é programar o objeto **butPesquisa** de maneira quase idêntica ao nosso antigo botão de pesquisa:

- Código para o botão **butPesquisa**, responsável pela ativação da pesquisa:

1. Dê um duplo clique no objeto:

```
procedure TF_SelCate.ButPesquisaClick(Sender: TObject);           Inicia o procedimento
begin
  if (EdtTrecho.Text = '') then                                    Verifica se foi digitado algo
  begin
    MessageDlg('Não foi especificado uma sigla!', mtError, [mbOK], 0);
    Exit;                                                         Abandona o procedimento
  end;
  try                                                             Ativa o comando Try
    DM_Modelo.TabCategor.FindNearest([EdtTrecho.Text]);          Faz a pesquisa na tabela
  except                                                         Caso aconteça alguma falha
    on exception do
      MessageDlg('Sigla especificada está inválida!', mtError, [mbOK], 0);
    end;
  end;                                                           Termina o procedimento
end;
```

U'O comando **Try** foi utilizado para prevermos qualquer possibilidade de erro durante a execução da pesquisa.

Enviando e recebendo variáveis

Existem duas maneiras de dois formulários trocarem valores de variáveis, ambas através da criação de variáveis definidas na seção **public**, a diferença está ou não na utilização do comando **property**.

No nosso antigo botão **Localizar**, enviamos para a função **InputQuery** o código da categoria corrente e a função nos devolve o código digitado, porque não fazemos o mesmo:

- Criando uma variável de escrita e leitura
 1. Vá para o *Code Editor*, e insira o trecho abaixo:

private	Parte de procedimentos privados
function GetTrecho: String ;	Função para enviar a variável
procedure SetTrecho(NewTrecho: String);	Procedimento para receber o trecho
public	Parte de procedimentos públicos
vCria: Boolean;	Cria uma variável pública
property CampTrecho: String read GetTrecho write SetTrecho;	Inicializa um variável
end ;	

U' O comando **Property** declara uma variável da seguinte maneira, a sub-opção **Read** envia um **string** através da função *GetTrecho*, e a sub-opção **Write** recebe o valor enviado através da variável para o procedimento *SetTrecho*.

2. Declarando o procedimento e a função:

function TF_SelCate.GetTrecho: String ;	Início da função
begin	
if vCria then	Verifica se o <i>DataModule</i> não existia
begin	
Result := DM_Modelo.TabCategorSIG_CATEG.Value;	Atribui ao resultado da função o valor do campo SIG_CATEG
DM_Modelo.Free;	
end ;	
end ;	
procedure TF_SelCate.SetTrecho(NewTrecho: String);	Início do procedimento
begin	
if vCria then	Se o <i>DataModule</i> não estiver criado
begin	
DM_Modelo := TDM_Modelo.Create(Application);	Cria o <i>DataModule</i>
DM_Modelo.TabCategor.FindKey([NewTrecho]);	Localiza o valor da string enviada
end ;	
end ;	

Alterando o formulário fCateg

Devemos agora mudar o código do formulário **F_Categ** para ativarmos a pesquisa, lembre-se que o *DataModule* é o mesmo e a tabela também, então não existe a necessidade de enviarmos ou trazermos uma variável com a sigla da Categoria, porém lembremos do botão de **cancelar**, precisamos então, caso o formulário não saia através do botão de **Ok**, retornamos ao registro anterior:

- Chame o formulário **F_Categ** através do objeto *Project Manager*, clique no objeto *butLocalizar*:

1. Dê um duplo clique no objeto:

```

procedure TF_Categ.ButLocalizarClick(Sender: TObject);           Aqui permanece o mesmo
var
  MarcaReg: TBookmark;                                           Cria uma variável de marca
begin
  if F_Menu.CriticaEdicao(DM_Modelo.DSCategor, 'Categoria', 'localizar' ) then
    Exit;

  with DM_Modelo.TabCateg do
    begin
      MarcaReg := GetBookmark;                                     Salva a posição do registro
      F_SelCate := TF_SelCate.Create(Application);               Cria o formulário de consulta
      F_SelCate.vCria := False;                                   Seta a variável pública
      if not F_SelCate.ShowModal = mrOk then                      Caso a saída do formulário não seja o botão OK
        GotoBookmark(MarcaReg);                                   Retorna ao registro marcado
      F_SelCate.Free;                                             Elimina o formulário de consulta
      FreeBookmark(MarcaReg);                                     Libera a variável de marca
    end;
end;

```

U'Os comandos de **BookMark** são todos aplicados a tabela por isso foi utilizado em conjunto com um comando **with** para a simplificação do código.

U'A variável declarada **objPesquisa** não é mais necessária.

U'Não é necessário criticar o comando **FindKey** pois o código retornado pela variável e pego diretamente do arquivo.

U'Não esqueça que o objeto **F_SelCate** faz parte da Unit **fSelCate** então é necessário fazer o uso desta Unit, para tanto altere o seguinte código (abaixo da diretiva de compilação):

*{\$R *.DFM}*

```

uses
  fMenu,                { Menu Principal do Sistema }
  DMModelo,             { Referencia ao DataModule }
  Dialogs,              { Utilizado para o controle da função MessageDlg }
  fSelCate;              { Seleciona Categoria }

```

Alterando o formulário fBasico

- Vamos finalmente tornar útil o objeto *butLocCateg* (botão para localizar a categoria).

1. Chame o formulário **F_Basico** e dê um duplo clique no objeto *butLocCateg*

```

procedure TF_Basico.ButLocCategClick(Sender: TObject);
begin
  if not (DM_Basico.DSBasico.State in [dsEdit, dsInsert]) then
    begin
      MessageDlg('Você não está no modo de edição!', mtInformation, [mbOK], 0);
    end;
  Exit;

```

```

end;

F_SelCate := TF_SelCate.Create(Application);           Cria o formulário de consulta
F_SelCate.vCria := True;                               Seta a variável pública
F_SelCate.CampTrecho := EditSIG_CATEG.Text;            Atribui a variável declarado o valor do
                                                         campo da tela (dispara a função SetTrecho)
if F_SelCate.ShowModal = mrOk then                    Executa o objeto F_SelCate
    EditSIG_CATEG.Text := F_SelCate.CampTrecho;        dispara o procedimento GetTrecho
F_SelCate.Free;                                       Elimina o formulário de consulta
end;

```

U' Note que para esse caso não queremos localizar uma categoria, mas sim atribuir ao objeto EditCOD_CATEG ao valor do F_SelCate.CampTrecho localizado

U' Não esqueça que o objeto **F_SelCate** faz parte da Unit **fSelCate** então é necessário fazer o uso desta Unit, para tanto altere o seguinte código (abaixo da diretiva de compilação):

```
{ $R *.DFM }
```

uses

```

DMBasico,      { Referência ao DataModule }
fMenu,         { Menu Principal do Sistema }
Dialogs,       { Gerente de Mensagens }
fCateg,        { Cadastro de Categorias }
fMusica,       { Cadastro de Músicas }
fSelCate;      { Seleciona Categoria }

```

- Salve o formulário, retire-o da área de formulários **auto-create**, execute e teste o projeto.
- Se alguma coisa deu errada, releia o capítulo, ou então confira o código para a **F_SelCate**:

```
unit fSelCate;
```

interface

uses

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
ExtCtrls, DBCtrls, Buttons, StdCtrls, Grids, DBGrids;
```

type

```

TF_SelCate = class(TForm)
    DBNavigator1: TDBNavigator;
    DBGrid1: TDBGrid;
    EdtTrecho: TMaskEdit;
    Label1: TLabel;
    ButOK: TBitBtn;
    ButCancela: TBitBtn;
    ButPesquisa: TSpeedButton;
    procedure ButPesquisaClick(Sender: TObject);

```

private

```

VeCria : boolean;
function GetTrecho: String;
procedure SetTrecho(NewTrecho: String);

```

public

```
property CampTrecho: String read GetTrecho write SetTrecho;
```



```
end;

var
  F_SelCate: TF_SelCate;

implementation

{$R *.DFM}

uses
  DMCateg; { Referência ao DataModule }

function TF_SelCate.GetTrecho: String;
begin
  if vCria then
  begin
    Result := DM_Modelo.TabCategorSIG_CATEG.Value;
    DM_Modelo.Free;
  end;
end;

procedure TF_SelCate.SetTrecho(NewTrecho: String);
begin
  if vCria then
  begin
    DM_Modelo := TDM_Modelo.Create(Application);
    DM_Modelo.TabCategor.FindKey([NewTrecho]);
  end;
end;

procedure TF_SelCate.ButPesquisaClick(Sender: TObject);
begin
  if (EdtTrecho.Text = '') then
  begin
    MessageDlg('Não foi especificado uma sigla!', mtError, [mbOK], 0);
    Exit;
  end;

  try
    DM_Modelo.TabCategor.FindNearest([EdtTrecho.Text]);
  except
    on exception do MessageDlg('Sigla especificada está inválida!', mtError, [mbOK], 0);
  end;
end;

end.
```

Comandos e suas funções, por ordem de aparição:

Property [variável]: [Tipo] **read** [função] **write** [procedimento] - Declara as propriedades para uma variável do tipo leitura (**read**) e do tipo escrita (**write**).

Result - constante utilizada para o retorno de uma função.

Try [declarações] - palavra reservada para marcar a primeira parte de um bloco de exceção.


Except [bloco de exceção] - caso alguma declaração dentro de um bloco de proteção **try** aconteça erro este bloco é executado.

On [tipo de exceção] **do** [declaração] - em conjunto com o bloco **try...except** define o código para executar um bloco de exceção.

Criando consultas para o cadastro

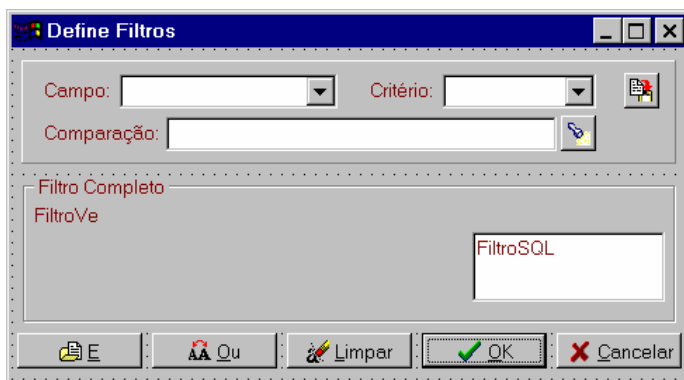
Acredito que você deve ter achado estranho o título das consultas do menu (CD's e Músicas por CD's), como já mencionei antes devemos permitir ao nosso usuário consultas ágeis pelos arquivos, senão não valerá a pena verificar no computador onde está determinada música, seria mais prático procurá-la manualmente.

Consultas SQL


O objeto *Query* , utilizado anteriormente para trabalhar como um contador, é mais poderoso do que se imagina, utilizando-o bem nos podemos dar poderes ao nosso usuário que ele jamais imaginou que fosse possível, e você verá que trabalhar com filtros pode-se transformar em uma agradável surpresa.

Realizando Consultas com Filtros

Os filtros são realizados para refinar determinados registros dentro de um ou mais arquivos, mostrarei de forma simples como utilizá-lo, observe o formulário a seguir:



Vamos criar um formulário idêntico a este que servirá para a montagem dos filtros:

- Inicialmente Clique no botão  (New Form) na *Speed Bar*, ou no menu principal a Clique no menu principal a opção **File** e **New...**, em *New Items*, na página *New* e clique no objeto entitulado *Form* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsDialog	Estilo da borda do objeto
Caption	Define Filtros	Label do objeto

Name	F_Filtro	Nome do objeto
Position	poScreenCenter	Posição do objeto

- Crie para esta nova janela os seguintes objetos:

1º Parte - montagem do filtro

Panel (encontrado na página *Standard*) - servirá para guardar o bloco de montagem do filtro, ajuste seu tamanho para caber os outros objetos e coloque branco para a propriedade **Caption**. Antes de criar os outros objetos sempre marque este objeto.

3 objetos Label (encontrados na página *Standard*) - altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	Campo, Critério e Comparação (respectivamente)	Nome do objeto
Font	MS Sans Serif, Negrito, 8, Castanho	Tipo de letra a ser mostrada no objeto

ComboBox (encontrado na página *Standard*) - que armazenará o nome dos campos do arquivo, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	CBCampo	Nome do objeto
Font	MS Sans Serif, Normal, 8, Azul Marinho	Tipo de letra a ser mostrada no objeto
Items	Nome; Tipo e Categoria (um em cada linha)	Conteúdo do objeto
Style	csDropDown	Estilo do objeto
Text		Texto a ser mostrado para o Combo quando nada for selecionado

ComboBox (encontrado na página *Standard*) - que armazenará os critérios de pesquisa, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	CBCriterio	Nome do objeto
Font	MS Sans Serif, Normal, 8, Azul Marinho	Tipo de letra a ser mostrada no objeto
Items	=; >; <; >=; <=; <> e Dentro de	Conteúdo do objeto (coloque um em cada linha)
Style	csDropDown	Estilo do objeto
Text		Texto a ser mostrado para o Combo quando nada for selecionado

Edit (encontrado na página *Standard*) - que armazenará o conteúdo a ser comparado, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	EditCompara	Nome do objeto
Font	MS Sans Serif, Normal, 8, Azul Marinho	Tipo de letra a ser mostrada no objeto
Text		Texto a ser mostrado para o campo

SpeedButton (encontrado na página *Additional*) - botão que confirmará a condição para o filtro escolhida, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\COPY.BMP	Imagem a ser mostrada no objeto
Hint	Confirma os dados digitados	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Name	ButEnvia	Nome do objeto

SpeedButton (encontrado na página *Additional*) - este botão servirá para chamar a tela de consulta a tabela, ficando originalmente invisível, sendo ativado caso o campo escolhido seja a “sigla da categoria”, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\LANTERNA.BMP	Imagem a ser mostrada no objeto (consulte o cap.6, criação do objeto ButLocCateg sobre a imagem)
Hint	Pesquisa Categoria	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Name	ButConsulta	Nome do objeto
Visible	False	Objeto deve ser mostrado

2º Parte - servirá para guardar o filtro montado e esconderá o filtro em linguagem pura SQL, ajuste seu tamanho para caber os outros objetos

GroupBox (encontrado na página *Standard*) - servirá para guardar os blocos montados do filtro, ajuste seu tamanho para caber os outros objetos, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	Filtro Completo	Label do objeto
Font	MS Sans Serif, Negrito, 8, Castanho	Tipo de letra a ser mostrada no objeto

Edit (encontrado na página *Standard*) - que mostrará ao usuário o filtro por ele criado, ele ocupará toda a área interna do objeto *GroupBox*, altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsNone	Estilo da borda
Color	clBtnFalse (mesma cor do objeto <i>GroupBox</i>)	Cor do objeto
Ctl3D	False	Possui desenho em 3D
Font	MS Sans Serif, Normal, 8, Castanho	Tipo de letra a ser mostrada no objeto
Name	FiltroVe	Nome do objeto
ReadOnly	True	Objeto apenas para leitura, não permitida alteração do seu conteúdo.

Edit (encontrado na página *Standard*)- que montará o nosso filtro em linguagem pura SQL, ele ficará invisível em modo de execução, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	FiltroSQL	Nome do objeto
Visible	False	Se o objeto ficará visível

3º Parte - botões de controle

BitBtn (encontrado na página *Additional*) - botão que servirá para cláusula “E - Tipo: *Um e Outro*”, ficará inicialmente desabilitado em modo de execução, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	ButE	Nome do objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\FOLDRDOC.BMP	Imagem a ser mostrada no objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Hint	Um “e” Outro	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Caption	&E	Label do Objeto
Enabled	False	Habilitação para o uso

BitBtn (encontrado na página *Additional*)- botão que servirá de “Ou - Tipo: *Um ou Outro*”, ficará inicialmente desabilitado em modo de execução, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	ButOu	Nome do objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\FONTBOLD.BMP	Imagem a ser mostrada no objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Hint	Um “ou” Outro	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Caption	O&u	Label do Objeto
Enabled	False	Habilitação para o uso

BitBtn (encontrado na página *Additional*) - botão que servirá para limpar o filtro, ficará inicialmente desabilitado em modo de execução, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	ButLimpar	Nome do objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\CLEAR.BMP	Imagem a ser mostrada no objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Hint	Apaga o filtro existente	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Caption	&Limpar	Label do Objeto
Enabled	False	Habilitação para o uso

BitBtn (encontrado na página *Additional*) - botão que servirá para confirmar o filtro, ele ficará inicialmente desabilitado em modo de execução, altere as seguintes propriedades:

Propriedade	Valor	Descrição
-------------	-------	-----------

Kind	bkOK	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption, Glyph e ModalResult
Name	ButOK	Nome do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Hint	Confirma o filtro editado	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Caption	&OK	Label do Objeto
Enabled	False	Habilitação para o uso

BitBtn (encontrado na página *Additional*) - botão que servirá para abandonar o filtro, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Kind	bkCANCEL	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption, Glyph e ModalResult
Name	ButCancela	Nome do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Hint	Cancela o filtro editado	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
Caption	&Cancelar	Label do Objeto

Programando o formulário

Antes de prosseguirmos salve o formulário com o nome de fFiltro.

O código para este objeto é simples requer apenas que você compreenda que o formulário deve fazer. Este formulário servirá de base para a montagem de um filtro que será mostrado por um *dbGrid* construído no segundo formulário, devemos devolver o código que completará a execução de um objeto *Query* colocado no segundo. Ex.: SELECT * FROM BASICO WHERE [condição criada no formulário] :

- Código para o botão **butEnvia**, quando for confirmado os dados digitados:

1. Dê um duplo clique no objeto:

```
procedure TF_Filtro.ButEnviaClick(Sender: TObject);
```

```
var
```

```
  NomeCampo : String;
```

Cria uma variável para armazenar o nome real do campo

```
begin
```

```
  case cbCampo.ItemIndex of
```

```
    0 { Nome } : NomeCampo:= 'NOM_DISCO';
```

Seleciona o nome do campo correto para o SQL

```
    1 { Tipo } : NomeCampo:= 'TIP_DISCO';
```

```
    2 { Categoria } : NomeCampo:= 'SIG_CATEG';
```

```
  end;
```

```
if (cbCritério.ItemIndex = 6) then
```

```
  FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo
```

se o Critério escolhido for “Dentro de” comando “LIKE” em SQL

```
  + ' ' + 'Like "%' + EditCompara.Text + '%"')'
```

```

else
    FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo
        + '' + cbCriterio.Text + '' + EditCompara.Text + ''');
    caso contrário atribui o valor
    escolhido ">, <, >=, <= e <>"

FiltroVe.Text := FiltroVe.Text + cbCampo.Text + '' +
    cbCriterio.Text + '' + EditCompara.Text;
    Acerta o filtro

cbCampo.ItemIndex := -1;
cbCriterio.ItemIndex := -1;
EditCompara.Text := '';
    Reseta os três campos envolvidos

cbCampo.Enabled := False;
cbCriterio.Enabled := False;
EditCompara.Enabled := False;
ButConsulta.Visible := False;
    Desabilita os campos de edição

ButE.Enabled := True;
ButOu.Enabled := True;
ButOk.Enabled := True;
ButLimpa.Enabled := True;
    Habilita os botões
end;

U' caso o sistema que você desenvolver tenha campos do tipo Data altere a linha do
critério para o seguinte:
if (cbCriterio.ItemIndex = 6) then
    if cbCampo.ItemIndex in [número do campos data] then
        FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo + '' +
            'Like "' +
            FormatDateTime('MM/DD/YY', StrToDate(EditCompara.Text))
            + '")'
        Critério "Dentro de"
        Vê os campos Data
        Condição para este tipo de cmp
    else
        FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo + '' +
            'Like "' + EditCompara.Text + '%"')
        Condição comum para o LIKE
    else
        if cbCampo.ItemIndex = in [número do campos data] then
            FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo + '' +
                cbCriterio.Text + '' +
                FormatDateTime('MM/DD/YY', StrToDate(EditCompara.Text)) + ''')'
            As outras condições
            Vê os campos Data
        else
            FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo + '' +
                cbCriterio.Text + '' + EditCompara.Text + ''');
            Condições para outros campos

```

U'...o resto ficará idêntico. Não esqueça no comando **USES** colocar a biblioteca **SysUtils** para a função **FormatDateTime**

- Código para o botão **butE**, quando será escolhida “e” uma outra condição:
 1. Dê um duplo clique no objeto:

```

procedure TF_Filtro.ButEClick(Sender: TObject);
begin
    FiltroVe.Text := FiltroVe.Text + ' E ';
    FiltroSQL.Text := FiltroSQL.Text + ' And ';
    Altera os filtros

    cbCampo.Enabled := True;
    cbCriterio.Enabled := True;
    Habilita os campos de edição

```

```
EditCompara.Enabled := True;  
ButLimpa.Enabled := True;
```

```
ButE.Enabled := False;           Desabilita os botões  
ButOu.Enabled := False;  
ButOk.Enabled := False;
```

```
end;
```

- Código para o botão **butOu**, quando será escolhida “ou” uma outra condição:

1. Dê um duplo clique no objeto:

```
procedure TF_Filtro.ButOuClick(Sender: TObject);
```

```
begin
```

```
FiltroVe.Text := FiltroVe.Text + ' Ou ';
```

Altera os filtros

```
FiltroSQL.Text := FiltroSQL.Text + ' Or ';
```

```
cbCampo.Enabled := True;
```

Habilita os campos de edição

```
cbCritério.Enabled := True;
```

```
EditCompara.Enabled := True;
```

```
ButLimpa.Enabled := True;
```

```
ButE.Enabled := False;
```

Desabilita os botões

```
ButOu.Enabled := False;
```

```
ButOk.Enabled := False;
```

```
end;
```

- Código para o botão **butLimpa**, quando for limpo os filtros:

1. Dê um duplo clique no objeto:

```
procedure TF_Filtro.ButLimpaClick(Sender: TObject);
```

```
begin
```

```
cbCampo.ItemIndex := -1;
```

Inicia todas as condições

```
cbCritério.ItemIndex := -1;
```

```
EditCompara.Text := '';
```

```
FiltroVe.Text := '';
```

```
FiltroSQL.Text := 'Select * from Basico where ';
```

```
cbCampo.Enabled := True;
```

Habilita os campos de edição

```
cbCritério.Enabled := True;
```

```
EditCompara.Enabled := True;
```

```
ButE.Enabled := False;
```

Desabilita os botões

```
ButOu.Enabled := False;
```

```
ButOk.Enabled := False;
```

```
ButLimpa.Enabled := False;
```

```
end;
```

- Quando o formulário for criado é iniciado o filtro:

1. Selecione o objeto **F_Filtro** na tela de eventos dê um duplo click sobre a opção OnCreate:

```
procedure TF_Filtro.FormCreate(Sender: TObject);
```

```
begin
```

```
FiltroVe.Text := '';
```

Limpa o filtro visível

```
FiltroSQL.Text := 'Select * from BASICO where ';
```

Inicia o filtro escondido

```
end;
```


- Caso o campo escolhido seja “Sigla da Categoria” é necessário habilitar o botão de consulta:
 1. Selecione o objeto **CBCampo** na tela de eventos dê um duplo clique na opção **OnChange**:

```

procedure TF_Filtro.CBCampoChange(Sender: TObject);
begin
    CBCriterio.SetFocus;                                Desvia o foco para o próximo campo
    if cbCampo.ItemIndex = 2 then                        Se o campo seleciona for SIG_CATEG
        ButConsulta.Visible := True                    Torna visível o objeto ButConsulta
    else                                                  Senão
        ButConsulta.Visible := False;                  Torna invisível o objeto ButConsulta
end;
    
```

- Código para o objeto **ButConsulta** que fará a consulta de categoria:
 1. Dê um duplo clique no objeto:

```

procedure TF_Filtro.ButConsultaClick(Sender: TObject);
begin
    if cbCampo.ItemIndex = 2 then                        Se o campo for SIG_CATEG
        begin
            F_SelCate.CampTrecho := EditCompara.Text;    Envia o campo
            if F_SelCate.Inicio = mrOk then              Verifica se foi retornado OK
                EditCompara.Text := F_SelCate.CampTrecho; Transfere o campo
        end;
end;
    
```

- Perfumaria para o objeto **CBCriterio** quando selecionado um critério será o controle transferido para o campo seguinte:
 1. Selecione o objeto na tela de eventos e dê um duplo clique na opção **OnChange**:

```

procedure TF_Filtro.CBCriterioChange(Sender: TObject);
begin
    EditCompara.SetFocus;                                Desvia o foco para o próximo campo
end;
    
```

Necessitamos agora criar uma variável comum que devolverá o código SQL para a tela secundária:

- Transfira para a janela do **Code Editor**:
 1. Após o comando **PRIVATE** digite:

```

private
    function GetSQL: TEdit;                                Inicia uma função Particular
public
    property SQLString : TEdit read GetSQL;                Inicia uma variável pública só de leitura
end;
    
```

```

var
    F_Filtro: TF_Filtro;
    
```

implementation

```

{$R *.DFM}
    
```

```

uses
    fSelCate;                                Utilização das bibliotecas
                                            Seleciona Categoria
    
```

```
function TF_Filtro.GetSQL: TEdit;
```

```
begin
```

```
  Result := FiltroSQL;
```

Envia como resultado para a variável o filtro escondido

```
end;
```

- Salve o formulário, teste o projeto.
- Se alguma coisa deu errada, releia o capítulo, ou então confira o código para a **F_Filtro**:

```
unit Ffiltro;
```

```
interface
```

```
uses WinTypes, WinProcs, Classes, Graphics, Forms,  
      Controls, Buttons, StdCtrls, ExtCtrls;
```

```
type
```

```
TF_Filtro = class(TForm)
```

```
  Panel1: TPanel;
```

```
  CBCampo: TComboBox;
```

```
  CBCriterio: TComboBox;
```

```
  EditCompara: TEdit;
```

```
  ButEnvia: TSpeedButton;
```

```
  GroupBox1: TGroupBox;
```

```
  Label1: TLabel;
```

```
  Label2: TLabel;
```

```
  Label3: TLabel;
```

```
  ButCancela: TBitBtn;
```

```
  ButE: TBitBtn;
```

```
  ButOu: TBitBtn;
```

```
  Label4: TLabel;
```

```
  FiltroVe: TEdit;
```

```
  FiltroSQL: TEdit;
```

```
  ButOk: TBitBtn;
```

```
  ButLimpa: TBitBtn;
```

```
  ButConsulta: TSpeedButton;
```

```
  procedure ButEnviaClick(Sender: TObject);
```

```
  procedure ButEClick(Sender: TObject);
```

```
  procedure ButOuClick(Sender: TObject);
```

```
  procedure ButLimpaClick(Sender: TObject);
```

```
  procedure FormCreate(Sender: TObject);
```

```
  procedure CBCampoChange(Sender: TObject);
```

```
  procedure ButConsultaClick(Sender: TObject);
```

```
  procedure CBCriterioChange(Sender: TObject);
```

```
private
```

```
  function GetSQL: TEdit;
```

```
public
```

```
  property SQLString : TEdit read GetSQL;
```

```
end;
```

```
var
```

```
  F_Filtro: TF_Filtro;
```

```
implementation
```

```
{ $R *.DFM }
```

```
uses
```

```
  fSelCate;
```

```
function TF_Filtro.GetSQL: TEdit;
```

```
begin
```

```
  Result := FiltroSQL;
```

```
end;
```

```
procedure TF_Filtro.ButEnviaClick(Sender: TObject);
```

```
var
```

```
  NomeCampo : String;
```

```
begin
```

```
  case cbCampo.ItemIndex of
```

```
    0 { Nome }      : NomeCampo:= 'NOM_DISCO';
```

```
    1 { Tipo }      : NomeCampo:= 'TIP_DISCO';
```

```
    2 { Categoria } : NomeCampo:= 'SIG_CATEG';
```

```
  end;
```

```
  if (cbCritério.ItemIndex = 6) then
```

```
    FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo  
      + ' ' + 'Like "%' + EditCompara.Text + '%" )'
```

```
  else
```

```
    FiltroSQL.Text := FiltroSQL.Text + '(' + NomeCampo  
      + ' ' + cbCritério.Text + ' ' + EditCompara.Text + ' ' );
```

```
  FiltroVe.Text := FiltroVe.Text + cbCampo.Text + ' ' +  
    cbCritério.Text + ' ' + EditCompara.Text;
```

```
  cbCampo.ItemIndex := -1;
```

```
  cbCritério.ItemIndex := -1;
```

```
  EditCompara.Text := '';
```

```
  cbCampo.Enabled := False;
```

```
  cbCritério.Enabled := False;
```

```
  EditCompara.Enabled := False;
```

```
  ButConsulta.Visible := False;
```

```
  ButE.Enabled := True;
```

```
  ButOu.Enabled := True;
```

```
  ButOk.Enabled := True;
```

```
  ButLimpa.Enabled := True;
```

```
end;
```

```
procedure TF_Filtro.ButEClick(Sender: TObject);
```

```
begin
```

```
  FiltroVe.Text := FiltroVe.Text + ' E ';
```

```
  FiltroSQL.Text := FiltroSQL.Text + ' And ';
```

```
  cbCampo.Enabled := True;
```

```
  cbCritério.Enabled := True;
```

```
  EditCompara.Enabled := True;
```

```
  ButLimpa.Enabled := True;
```

```
  ButE.Enabled := False;
```

```
    ButOu.Enabled := False;
    ButOk.Enabled := False;
end;

procedure TF_Filtro.ButOuClick(Sender: TObject);
begin
    FiltroVe.Text := FiltroVe.Text + ' Ou ';
    FiltroSQL.Text := FiltroSQL.Text + ' Or ';

    cbCampo.Enabled := True;
    cbCritério.Enabled := True;
    EditCompara.Enabled := True;
    ButLimpa.Enabled := True;

    ButE.Enabled := False;
    ButOu.Enabled := False;
    ButOk.Enabled := False;
end;

procedure TF_Filtro.ButLimpaClick(Sender: TObject);
begin
    cbCampo.ItemIndex := -1;
    cbCritério.ItemIndex := -1;
    EditCompara.Text := '';
    FiltroVe.Text := '';
    FiltroSQL.Text := 'Select * from BASICO where ';

    cbCampo.Enabled := True;
    cbCritério.Enabled := True;
    EditCompara.Enabled := True;

    ButE.Enabled := False;
    ButOu.Enabled := False;
    ButOk.Enabled := False;
    ButLimpa.Enabled := False;
end;

procedure TF_Filtro.FormCreate(Sender: TObject);
begin
    FiltroVe.Text := '';
    FiltroSQL.Text := 'Select * from Basico where ';
end;

procedure TF_Filtro.CBCampoChange(Sender: TObject);
begin
    CBCritério.SetFocus;
    if cbCampo.ItemIndex = 2 then
        ButConsulta.Visible := True
    else
        ButConsulta.Visible := False;
end;

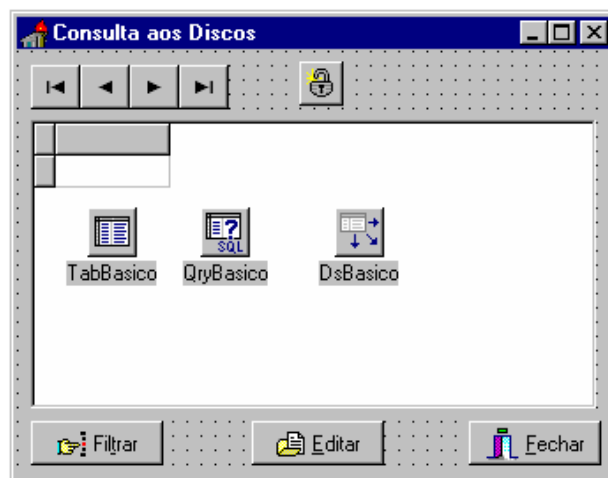
procedure TF_Filtro.ButConsultaClick(Sender: TObject);
begin
    if cbCampo.ItemIndex = 2 then
        begin
```

```
F_SelCate.CampTrecho := EditCompara.Text;  
if F_SelCate.Inicio = mrOk then  
    EditCompara.Text := F_SelCate.CampTrecho;  
end;  
end;  
  
procedure TF_Filtro.CBCriterioChange(Sender: TObject);  
begin  
    EditCompara.SetFocus;  
end;  
  
end.
```


U'Não foi colocado nenhum comando que já não tenha sido visto anteriormente

Criando o formulário Gerente do Filtro

Criado o formulário anterior, a idéia agora é a seguinte, criaremos um outro formulário que conterá um objeto *dbGrid* para mostrar todos os campos de determinada tabela, quando for clicado o botão filtrar, será disparado o formulário anterior, ao retornar o usuário poderá ativar/desativar o filtro através do *SpeedButton* (cadeado), ou ainda será possível editar um registro selecionado :



Vamos criar um formulário idêntico a este que servirá para o gerenciamento do filtro:

- Inicialmente Clique no botão  (New Form) na *Speed Bar*, ou no menu principal a Clique no menu principal a opção **File** e **New...**, em *New Items*, na página *New* e clique no objeto entitulado *Form* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsDialog	Estilo da borda do objeto
Caption	Consulta aos Discos	Label do objeto
Name	F_ConDisc	Nome do objeto

Position poScreenCenter Posição do objeto

- Crie para esta nova janela os seguintes objetos:

Table (encontrado na página *Data Access*) - Tabela que se alternará com a query, altere as seguintes propriedades:

Propriedade	Valor	Descrição
DatabaseName	AliasDisco	Nome do Alias ou a localização do diretório das tabelas
TableName	BASICO.DBF	Nome externo da tabela
Name	TabBasico	Nome do objeto
IndexFieldNames	NOM_DISCO	Nome do campo indexado

DataSource (encontrado na página *Data Access*) - Posicione-o ao lado do objeto *TabBasico*. , altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSet	TabBasico	Nome da tabela vinculada
Name	DSBasico	Nome do objeto

Query (encontrado na página *Data Access*) - Query que se alternará com a tabela, altere as seguintes propriedades:

Propriedade	Valor	Descrição
DatabaseName	AliasDisco	Nome do Alias ou a localização do diretório das tabelas
SQL	SELECT * FROM BASICO	Comando SQL para consulta
Name	QryBasico	Nome do objeto

DBNavigator (encontrado na página *Data Controls*) - Barra de navegação que servirá para andar sobre os registro, lembre-se não deve ser permitida manutenção nos mesmos:

Propriedade	Valor	Descrição
DataSource	DSBasico	DataSource associado
VisibleButtons	[nbFirst, nbPrior, nbNext, nbLast]	Botões visíveis
Hints	Primeiro; Anterior; Próximo; Último	Ajuda on-line que será mostrado sobre cada botão (coloque um em cada linha)
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line

SpeedButton (encontrado na página *Additional*) - botão que servirá para ativar ou desativar o filtro, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Name	ButAtivaDesativa	Nome do objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\LOCKOPEN.BMP	Imagem a ser mostrada no objeto
Hint	Ativa/Desativa o filtro	Ajuda on-line para o objeto específico
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line
GroupIndex	1	Pertence a um grupo de botões, esta propriedade faz com que o botão permaneça em estado de pressionamento.

AllowAllUp	True	Permite que um grupo de botões não tenha nenhum botão em estado de pressionamento.
------------	------	--

BitBtn (encontrado na página *Additional*) - botão que servirá para montagem do filtro, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	Fil&trar	Label do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\FIND.BMP	Imagem a ser mostrada no objeto
Hint	Montagem do Filtro	Ajuda on-line para o objeto específico
Name	ButFiltrar	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line

BitBtn (encontrado na página *Additional*) - botão que servirá para editar o campo selecionado no grid, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Caption	&Editar	Label do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Glyph	[DiretórioDelphi]\IMAGES\BUTTONS\FOLDRDOC.BMP	Imagem a ser mostrada no objeto
Hint	Edita campo selecionado	Ajuda on-line para o objeto específico
Name	ButEditar	Nome do objeto
ShowHint	True	Mostrar o conteúdo da propriedade hint sob a forma de uma caixa de ajuda on-line

BitBtn (encontrado na página *Additional*) - botão que servirá para editar o filtro, altere as seguintes propriedades:

Propriedade	Valor	Descrição
Kind	bkClose	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption, Glyph e ModalResult
Name	ButFechar	Nome do objeto
Font	MS Sans Serif, Normal, 8, Preto	Tipo de letra a ser mostrada no objeto
Caption	&Fechar	Label do Objeto

dbGrid (encontrado na página *Data Controls*) - Objeto Grid que mostrará os campos, altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSource	DSDocum	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption, Glyph e ModalResult
Name	dbGridBasico	Nome do objeto
Font	MS Sans Serif, Normal, 8, Azul Marinho	Tipo de letra a ser mostrada para os campos
Options	dgTitles, dgIndicator, dgColLines, dgRowLines, dgRowSelect	Opções do Grid
TitleFont	MS Sans Serif, Negrito, 8, Castanho	Tipo de letra a ser mostrada para o título dos campos

Programando o formulário

Antes de prosseguirmos salve o formulário com o nome de fConDisc.

Este formulário mostrará os dados filtrados no objeto *dbGrid*, para filtrar os dados utilizaremos o objeto *Query*, tudo o que temos a fazer é controlar quando o filtro está ativo (objeto *DataSource* aponta para o objeto *Query*) ou quando o filtro não está ativo (objeto *DataSource* aponta para o objeto *tTable*)

- Código para quando o formulário é iniciado, abertura das tabelas:
 1. Vire para o **CodeEditor** e insira o seguinte código abaixo:

```
private
  { Private declarations }
public
  procedure Inicio;                      Declara um procedimento público
end;

var
  F_ConDisc: TF_ConDisc;

implementation

{$R *.DFM}

uses
  fFiltro,                             Utiliza a Unit para a montagem do filtro
  fEdtDisc;                             Utiliza a Unit para a edição do registro

procedure TF_ConDisc.Inicio;
begin
  TabBasico.Open;                      Inicia a tabela
  ShowModal;                           Mostra o formulário em modo modal
end;

• Código para o botão Filtrar, quando for solicitada a edição do filtro.
  1. Dê um duplo clique no objeto butFiltrar:

procedure TF_ConDisc.ButFiltrarClick(Sender: TObject);
begin
  if ButAtivaDesativa.Down then        Verifica se o botão de prender o filtro
  begin                                está pressionado.
    MessageDlg('Desative a filtragem antes de editar os filtros...',
      mtInformation, [mbOK], 0);
    exit;
  end;

  if F_Filtro.ShowModal = mrOk then     Chama o filtro e se for retornado OK
  begin
    QryBasico.SQL.Clear;                Limpa a Query
    QryBasico.SQL.Add(F_Filtro.SQLString.Text);  Adiciona o novo filtro na Query
  end;
end;
```

- Código para o botão Editar, quando for marcado um registro para visualização geral.

1. Dê um duplo clique no objeto **butEditar**:

```

procedure TF_ConDisc.ButEditarClick(Sender: TObject);
begin
  if ButAtivaDesativa.Down then           Verifica se existe filtro preso
    F_EdtDisc.Inicio(QryBasicoCOD_DISCO.Value) Chama a edição através da Query
  else
    F_EdtDisc.Inicio(TabBasicoCOD_DISCO.Value); Chama a edição através da tTable
end;

```

- Código para o botão Ativa ou Desativa o filtro, para prender ou liberar o filtro.

1. Dê um duplo clique no objeto **butAtivaDesativa**:

```

procedure TF_ConDisc.ButAtivaDesativaClick(Sender: TObject);
begin
  if not ButAtivaDesativa.Down then       Se o filtro não estiver ativo
    DSBasico.Dataset := TabBasico          Coloca o DataSource apontado para o tTable
  else
    try                                    Cláusula para um bloco de exceção
      QryBasico.Close;                     Desativa a Query
      QryBasico.Open;                     Ativa a Query
      DSBasico.Dataset := QryBasico;       Coloca o DataSource apontado para a Query
    except                                Caso de alguma coisa errada com o filtro
      DSBasico.Dataset := TabBasico;       Coloca o DataSource apontado para o tTable
      raise;                               Complemento do comando Try...except...raise
    end;
end;

```

- Salve o formulário.
- Com certeza será mostrado um erro para a chamada da unit **F_EdtDisc**.

Editando os registros

O formulário que editará os registro não será montado passo a passo, pois é muito simples de ser construído, ao invés disto darei apenas algumas dicas a respeito da criação do mesmo:

1. Copie o formulário **F_Basico**, renomeando-o para **F_EdtDisc**;
2. Remova os objetos **dbNavigator**, **Query** e as tabelas **tabCategConf** e **tabBasicoConf**, das tabelas restante, entre no **FieldsEditor** de cada uma e remova os campos;
3. Retire todo o código e proteja as tabelas com a opção *ReadOnly = True*;
4. Retire todos os botões, deixando apenas o botão para Fechar o formulário;
5. Insira os seguintes procedimentos:

a. Procedure INICIO, nas declarações **public** (procedure Inicio(var CodDisco: Integer);)

```

procedure TF_EdtDisc.Inicio(var CodDisco: Integer);
begin
  TabBasico.Open;           Abre as tabelas
  TabMusica.Open;
  TabCateg.Open;
  TabBasico.FindKey(CodDisco); Posiciona a tabela Basico no CodDisco enviado
end;

```

```
ShowModal;  
end;
```

b. Procedure FormClose, no evento **OnClose** do objeto F_EdtDisc

```
procedure TF_EdtDisc.FormClose(Sender: TObject; var Action: TCloseAction);  
begin  
    TabBasico.Close;           Fecha as tabelas  
    TabMusica.Close;  
    TabCateg.Close;  
end;
```

6. Salve e compile o formulário com o nome **fEdtDisc**.

- Agora sim, você pode compilar o projeto e rodar o projeto, para uma conferência final aí vai o código completo do formulário **F_ConDisc**.

```
unit Fcondisc;
```

```
interface
```

```
uses
```

```
WinTypes, WinProcs, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, DB, DBTables, Grids, DBGrids, StdCtrls, Buttons,  
ExtCtrls, DBCtrls;
```

```
type
```

```
TF_ConDisc = class(TForm)  
    DBGrid1: TDBGrid;  
    DSBasico: TDataSource;  
    ButFiltrar: TBitBtn;  
    ButFechar: TBitBtn;  
    QryBasico: TQuery;  
    DBNavigator1: TDBNavigator;  
    TabBasico: TTable;  
    ButEditar: TBitBtn;  
    ButAtivaDesativa: TSpeedButton;  
    TabBasicoCOD_DISCO: TSmallintField;  
    TabBasicoNOM_DISCO: TStringField;  
    TabBasicoTIP_DISCO: TStringField;  
    TabBasicoCOD_CATEG: TStringField;  
    QryBasicoCOD_DISCO: TSmallintField;  
    QryBasicoNOM_DISCO: TStringField;  
    QryBasicoTIP_DISCO: TStringField;  
    QryBasicoCOD_CATEG: TStringField;  
    procedure ButFiltrarClick(Sender: TObject);  
    procedure ButEditarClick(Sender: TObject);  
    procedure ButAtivaDesativaClick(Sender: TObject);  
private  
    { Private declarations }  
public  
    procedure Inicio;  
end;
```

```
var
```

```
F_ConDisc: TF_ConDisc;
```

implementation

```
{ $R *.DFM }
```

uses

```
fFiltro,    { Chama a montagem do Filtro }  
fEdtDisc;  { Edita o Disco }
```

```
procedure TF_ConDisc.Inicio;
```

begin

```
  TabBasico.Open;  
  ShowModal;
```

```
end;
```

```
procedure TF_ConDisc.ButFiltrarClick(Sender: TObject);
```

begin

```
  if ButAtivaDesativa.Down then
```

begin

```
      MessageDlg('Desative a filtragem antes de editar os filtros...', mtInformation, [mbOK], 0);  
      exit;
```

```
    end;
```

```
  if F_Filtro.ShowModal = mrOk then
```

begin

```
      QryBasico.SQL.Clear;  
      QryBasico.SQL.Add(F_Filtro.SQLString.Text);
```

```
    end;
```

```
end;
```

```
procedure TF_ConDisc.ButEditarClick(Sender: TObject);
```

begin

```
  if ButAtivaDesativa.Down then
```

```
    F_EdtDisc.Inicio(QryBasicoCOD_DISCO.Value)
```

```
  else
```

```
    F_EdtDisc.Inicio(TabBasicoCOD_DISCO.Value);
```

```
end;
```

```
procedure TF_ConDisc.ButAtivaDesativaClick(Sender: TObject);
```

begin

```
  if not ButAtivaDesativa.Down then
```

```
    DSBasico.Dataset := TabBasico
```

```
  else
```

```
    try
```

```
      QryBasico.Close;  
      QryBasico.Open;  
      DSBasico.Dataset := QryBasico;
```

```
    except
```

```
      DSBasico.Dataset := TabBasico;
```

```
      raise;
```

```
    end;
```

```
end;
```

```
end.
```

Capítulo VIII

Relatórios

A parte impressa do *Delphi* fica a critério de três métodos: utilizar a ferramenta **ReportSmith**, com a unit **Printer** ou com a utilização da ferramenta **QuickReport** (disponível no *Delphi 1.0* a partir de compra de terceiro, mas disponível gratuitamente com o *Delphi 2.0*). O *Delphi* oferece um gerador de relatórios bastante poderoso, chamado **ReportSmith**. Com este utilitário, você pode criar relatórios associados aos diversos bancos que o *Delphi* se comunica.

Trabalhando com o ReportSmith

O **ReportSmith** possui alguns relatórios padrões, tais como **relatórios colunados** (mostra os dados em formas de colunas), **referência cruzada entre tabelas** (mostra os dados tipos uma planilha, associando duas ou mais tabelas), **etiquetas** (gera etiquetas em formatos padrão), **relatórios em modo de página** (mostra os registros de uma determinada tabela um por página).

Através do ReportSmith, é possível gerar relatórios com as seguintes características:


- Combinação de dados entre várias tabelas;
- Ordem e grupo livre de dados através de determinados campos;
- Inserção de cabeçalhos e rodapés;
- Funções de soma, média, máximo e mínimo;
- Criação de caixas de diálogo para inserção de informações, filtragem dos dados;
- Criação de relatórios do tipo master/details;
- Execução de macros durante o relatório; e
- Acesso as mais variadas bases de dados, entre elas:

Bases de Dados que o ReportSmith permite acesso		
Access	Informix	SQLBase
Btrieve	InterBase	Sybase
DB2	ORACLE	Teradata
dBase	Paradox	Arquivo .TXT
Excel	Raima	Unify
FoxPro	AS/400	Watcom SQL
INGRES	SQL Server	qualquer base com driver ODBC

O **ReportSmith** é um produto separado do *Delphi*, mas ao comprar o *Delphi*, você adquiriu automaticamente a licença para usar o **ReportSmith** bem como a distribuição livre do seu Run-Time.

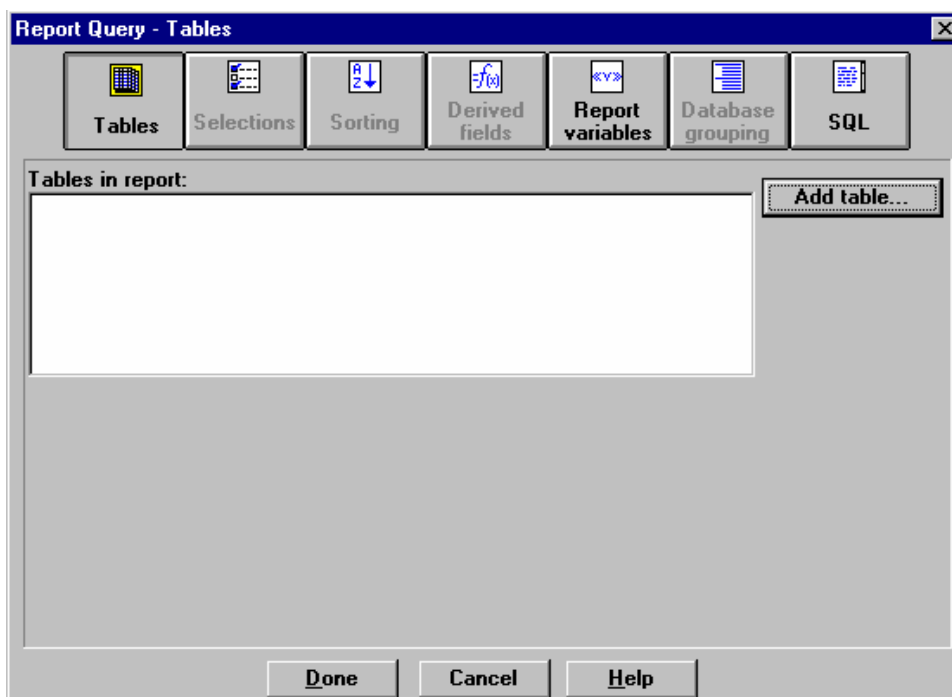
Criando relatório com o ReportSmith



Inicie o **ReportSmith** através do ícone , ou da opção do menu principal do *Delphi*, **Tools | ReportSmith**, dependendo da configuração da seção **Options**, será mostrada a tela para o início de um novo relatório.

Cancele quaisquer opção que apareça e ative no menu principal **File | Connections...**, lembre-se do **Alias** pois aqui precisamos criar uma conexão para a nossa base de dados, pressione o botão **New** e informe **Name**: DiscoAPP, **Type**: DBASE(IDAPI) e **Data File Path**: C:\SISTEMA\CADDISCO, pressione o botão **Save** e **OK**.

No menu principal opção **File | New...**, escolha a opção **Columnar Report**, clique no botão **Style** e escolha o estilo do relatório que mais lhe agrade e caso você deseje pode marcar o estilo escolhido como default através da opção **Use As Default** clique no botão **OK**, e clique no botão **OK** e a seguinte tela será mostrada:



Tables - Adiciona, remove ou altera tabelas ou cria links entre elas.

Selections - Cria, edita e exclui critérios de seleção.

Sorting - Adiciona, remove ou reordena dados a serem organizados.

Derived fields - Cria e organiza dados calculados para outras colunas do relatório, baseado em comandos SQL ou na linguagem de macro ReportBasic.

Report variables - Cria, edita ou exclui variáveis do relatório. Estas variáveis podem ser criadas para atender a critérios da cláusula **Selections** servindo de comunicação entre o **ReportSmith** e o **Delphi**.

Database grouping - Grupos de dados e critérios de seleção.

SQL - Permite a edição de declaração SQL para o relatório corrente.

1. Pressione o botão **Tables**.
2. Na caixa de diálogo **Tables**, pressione o botão **Add table...**
3. No combo **Connections**: selecione a conexão **DiscoAPP** e selecione a tabela **Basico.DB** e pressione o botão **OK**, repita a operação para tabela **Categor.DB**, confirme e retorne a tela anterior.
4. Pressione o botão **Add new link...**
5. O relacionamento entre **Basico** e **Categor** é através do campo **SIG_CATEG**, marque a opção **Include unmatched records** (inclua os registros não correspondentes) do lado de **BASICOxDB**, confirme e retorne a tela anterior.
6. Selecione a tabela **BASICOxDB** e pressione o botão **Table columns...**
7. Selecione o campo **FOT_CAPA** e marque a opção **Exclude from report** (retire do relatório), confirme e retorne a tela anterior.
8. Pressione o botão **Report variables** e crie as variáveis: **CodInicial** e **CodFinal** ambas com a opção **Type** igual a **Number** e a opção **Entry** igual a **Type-in**.
9. Pressione o botão **Selections** e clique no botão amarelo marcado com **1**. e escolha a opção **Add selection criteria** e marque o seguinte critério: **data field BASICOxDB.COD_DISCO is between report variable CodInicial and report variable CodFinal**.
10. Clique no botão **Sorting**, marque o campo **COD_DISCO** e clique no botão **Insert into sort list** (insira para a lista de ordenação).
11. Clique no botão **SQL** e compare a declaração criada:

```
SELECT
'BASICOxDB'.COD_DISCO, 'BASICOxDB'.NOM_DISCO, 'BASICOxDB'.TIP_DISCO,
'BASICOxDB'.SIG_CATEG, 'CATEGORxDB'.SIG_CATEG, 'CATEGORxDB'.DES_CATEG'
FROM
'C:\SISTEMA\CADDISCO\BASICO.DB' BASICOxDB LEFT JOIN
'C:\SISTEMA\CADDISCO\CATEGOR.DB' CATEGORxDB ON ('BASICOxDB'.SIG_CATEG' =
'CATEGORxDB'.SIG_CATEG')
WHERE
((( 'BASICOxDB'.COD_DISCO BETWEEN <<CodInicial>> AND <<CodFinal>>)))
ORDER BY
'BASICOxDB'.COD_DISCO
```

12. Confirme o relatório pressionando o botão **Done**. Informe os dados iniciais e finais e aguarde a geração do relatório.

Organizando os campos do relatório

Acredito que a esta altura você tem um belo início de relatório nas mãos, agora teremos que alterar alguns campos que não ficaram direito:


1. Selecione os campos SIG_CATEG (tanto o título quanto o detalhe) e pressione **DEL**.
2. Selecione o detalhe do campo NOM_DISCO e pressione o botão direito do mouse, selecione a opção **Field Height** e marque a opção **Can Grow** (redimensione a variável conforme o tamanho) e **Can Shrink** (corte os caracteres não imprimíveis).
3. Troque os títulos de cada campo para: Código, Nome, Tipo e Categoria. (basta clicar duas vezes sobre cada título).
4. Troque o título do relatório para: **Cadastro de CD's**.
5. No menu selecione a opção **Insert | Field...**, selecione **System Fields**, e selecione o campo **Print Date/Time**, pressione o botão **Insert** e clique em qualquer posição da seção **Header**.
6. Compare agora o relatório gerado:

Cadastro de CD's			
Página 1 de 1		25.03.1998 02:06:31	
Código	Nome	Tipo	Categoria
1	Chico em Cy - Quarteto em Cy	DDD	Cantores Nacionais
2	João Bosco - Acústico	DDD	Cantores Nacionais
3	The Division Bell - Pink Floyd	DDD	Cantores Internacionais
4	Alma Gêmea - Fábio Jr.	DDD	Cantores Nacionais
5	The Lost Boys (Os garotos perdidos)	DDD	Temas de Filmes
6	The Platters - Smoke Gets In Your Eyes	DDD	Cantores Internacionais
7	The Best of James Bond 30th Anniversary Collection	DDD	Temas de Filmes

Salve o relatório gerado com o nome de RCADAST.RPT

Associando o relatório ao aplicativo


A parte difícil já foi realizada agora resta chamar o relatório através do nosso menu principal.

Chame o formulário fmenu e insira o objeto Report , encontrado na *Component Palette* página *Dialogs*, este objeto realiza o trabalho de configuração da impressora.

1. Clique na opção "Configura Impressora" chamando o evento `onClick`;
2. Insira o comando

```
procedure TFMenu.ConfuraImpressora1Click(Sender: TObject);
begin
  PrinterSetupDialog1.Execute;
end;
```

Criaremos agora uma janela simples de diálogo, onde selecionaremos um código inicial e final para os códigos do CD.

- Inicialmente Clique no botão  (New Form) na *Speed Bar*, ou no menu principal a Clique no menu principal a opção **F**ile e **N**ew..., em *New Items*, na página *New* e clique no objeto entitulado *Form* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsDialog	Estilo da borda do objeto
Caption	Imprime Cadastro Geral	Label do objeto
Name	F_DgGeral	Nome do objeto
Position	poScreenCenter	Posição do objeto

- Crie para esta nova janela os seguintes objetos:

Label (Localizado na página **Standard**) - crie três objetos labels:

Propriedade	Valor	Descrição
Caption	[1]. Informe o Código dos CD's a imprimir; [2]. Inicial: e [3]. Final:	Label do objeto
Font	[1]. MS Sans Serif, Negrito, 8, Azul Marinho e [2,3]. MS Sans Serif, Negrito, 8, Castanho	Tipo de letra a ser mostrada no objeto

BitBtn (Localizado na página **Additional**) - crie dois botões para confirmar ou cancelar a emissão do relatório:

Propriedade	Valor	Descrição
Kind	bkOk e bkCancel	Determina a classe a ser utilizada pelo objeto, automaticamente serão alteradas as propriedades: Caption , Glyph e ModalResult
Caption	&OK e &Cancelar	Label do objeto

Edit (Localizado na página **Standard**) - dois objetos de edição para inserção do código inicial e final:

Propriedade	Valor	Descrição
Text		Texto a ser apresentado inicialmente para o objeto
Font	MS Sans Serif, Normal, 8, Azul Marinho	Tipo de letra a ser mostrada no objeto

Compare com o formulário abaixo:



Programando o formulário

Antes de prosseguirmos salve o formulário com o nome de fDgGeral.

Para selecionarmos os códigos inicial e final, precisamos fazer uma pequena programação no formulário:

- Código para criar os dois campos que enviarão os códigos (lembra-se do capítulo VII - Referente a Consultas):
 1. Alterne para o **CodeEditor** e insira o seguinte código abaixo:

```
private
  function GetCodInicial: String;           Inicializa as funções
  function GetCodFinal: String;
public
  property CampInicial: String read GetCodInicial;  Cria tipo caractere as variáveis,
  property CampFinal: String read GetCodFinal;     apenas como saída
end;

var
  F_DgGeral: TF_DgGeral;

implementation

{$R *.DFM}

function TF_DgGeral.GetCodInicial: String;
begin
  Result := Edit1.Text;                    Envia como resultado o conteúdo do
end;                                       objeto Edit1

function TF_DgGeral.GetCodFinal: String;
begin
  Result := Edit2.Text;                    Envia como resultado o conteúdo do
end;                                       objeto Edit2
```



Por incrível que pareça isto é tudo, agora chame o objeto fMenu, insira o objeto Report encontrado na *Component Palette* página *Data Access*, este objeto realiza o trabalho de configuração da impressão.

1. Clique na opção “Relatório | Geral” chamando o evento onClick;

2. Insira o comando

```
procedure TF_Menu.Geral1Click(Sender: TObject);
begin
  if F_DgGeral.ShowModal = mrOK then           Chama e verifica se a DgGeral retornou OK
  with Report1 do                               Para o objeto Report1...
  begin
    ReportName := 'RCADAST.RPT';               Altera a propriedade Nome do Relatório
    InitialValues.Clear;                       Elimina os valores iniciais
    InitialValues.Add('@CodInicial=<'+F_DgGeral.CampInicial+'>'); Seta o valor CodInicial do
                                                relatório com o valor do CampInicial do formulário fDgGeral
    InitialValues.Add('@CodFinal=<'+F_DgGeral.CampFinal+'>');   Seta o valor CodFinal do
                                                relatório com o valor do CampFinal do formulário fDgGeral
    Run;                                       Inicia o relatório
  end;
end;
```

Salve e execute o sistema, informe trechos diferentes para o relatório.

U' Caso você queira que o ReportSmith não execute imediatamente o relatório mostrando antes uma prévia na tela modifique a propriedade do objeto **Report1** - *Preview* para *True*.

U' Você descobrirá que o **ReportSmith** é uma poderosa e simples ferramenta para a concepção de relatórios, o único problema seria com relatórios que se precise imprimir uma única ou poucas folhas, como um recibo ou um formulário pré-impresso, você descobrirá que ele é lento para estas tarefas, existem duas soluções para este caso usar a biblioteca *Printers* ou usar a impressão livre do formulário como veremos a seguir.

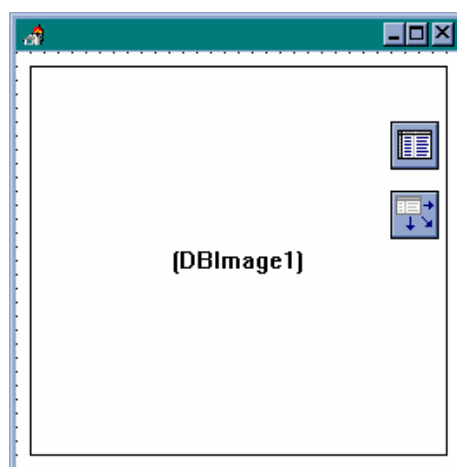
Imprimindo através do Formulário


Para o nosso próximo relatório, precisamos apenas imprimir a capa de um CD, se você relembra a foto foi armazenada em um campo do tipo BLOB chamado FOT_CAPA, no arquivo BASICO.

A idéia é simples, construir um formulário em branco, sem borda, este será chamado através do menu principal e solicitado a informação do código do CD a imprimir, neste formulário terá os objetos de acesso a Tabela (*Table* e *DataSource*) associado ao objeto DBImage



, encontrado na *Component Palette* na página *Data Controls*, conforme a figura abaixo:



- Inicialmente Clique no botão  (New Form) na *Speed Bar*, ou no menu principal a Clique no menu principal a opção **F**ile e **N**ew..., em *New Items*, na página *New* e clique no objeto entitulado *Form* e altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsNone	Estilo da borda do formulário
Caption		Label do objeto (Tarja azul do formulário)
Name	F_Capa	Nome do objeto
Position	poScreenCenter	Posição da janela (centralizado)
BorderIcons	[]	Elimine todos os botões da janela
Color	clWhite	Cor de fundo

- Crie para esta nova janela os seguintes objetos:

Table (Localizado na página **Data Access**) , altere as seguintes propriedades:

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Alias ou a localização do diretório das tabelas
TableName	BASICO.DB	Nome externo da tabela
IndexFieldNames	COD_DISCO	Nome do campo indexado
Name	TabBasico	Nome do objeto
ReadOnly	True	Somente para leitura

DataSource (Localizado na página **Data Access**) , altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSet	TabBasico	Nome da tabela vinculada
Name	DSBasico	Nome do objeto

dbImage (Localizado na página **Data Controls**) , altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSource	DSBasico	Nome do DataSource vinculado
DataField	FOT_CAPA	Nome do campo na tabela
Height	457	Largura

Stretch	True	Se o tamanho da imagem deve acompanhar o tamanho do objeto
Width	481	Altura do objeto

U' O tamanho da imagem foi colocada em 457 x 481 pois em impressoras padrão Epson este, após impresso, é o tamanho da capa do CD. É necessário que você faça os ajustes necessários para se adaptar ao padrão de sua impressora.

U' Cresça o tamanho do formulário em conjunto ao tamanho do objeto imagem.

Salve o formulário com o nome de fCapa.

Criando o Código

O código para este formulário e sua chamada a partir do menu principal é bem mais simples que o realizado anteriormente, verifique:

- Para o menu principal
 - Código para criar a chamada:
 1. Chame o formulário fMenu, e clique na opção “Capa do CD” :

```
procedure TF_Menu.CapadoCD1Click(Sender: TObject);
var
  NumCD: String;
begin
  NumCD := InputBox('Informe', 'Entre com o código do CD para imprimir:', '');
  if NumCD = '' then
  begin
    MessageDlg('Não foi informado um código para a impressão', mtInformation, [mbOk], 0);
    exit;
  end;
  F_Capa.Inicio(NumCD);
end;
```

U' O código para este evento é bem simples: cria uma variável caracter e solicita a entrada de seu valor através da função **InputBox**, caso não seja retornado nenhum valor cancela a impressão ao contrário chama o procedimento “inicio” do formulário “F_Capa” enviando o valor informado.

U' Não se esqueça de colocar a unidade fCapa no comando **Uses**.

- Para o formulário F_Capa
 - Código para solicitar e permitir a impressão:
 1. Chame o formulário F_Capa, e alterne para o modo do **Code Editor** :

```
procedure TF_Capa.Inicio(Numero: String);
begin
  TabBasico.Open;
  if not TabBasico.FindKey([StrToFloat(Numero)]) then
    MessageDlg('Código do CD inexistente !', mtError, [mbOk], 0)
```

U' Não se esqueça de declarar o procedimento nas declarações **Public. (procedure**
Inicio(Numero: String);)

etc. Alguns tipos de bandas são impressas automaticamente, enquanto que outros tipos necessitam dos objetos **QRGroup** ou **QRDetailLink** para seu controle. É possível também utilizar-se de múltiplas bandas de um mesmo tipo.

- QRGroup



Os grupos podem ser formados com o auxílio deste componente, se utiliza de cabeçalho e rodapé para o controle do grupo. É possível criar no máximo 10 níveis (a propriedade *Level* varia de 0 a 9) sendo que os níveis mais baixos dominam a impressão dos mais altos.

- QRDetailLink



O componente QRDetailLink é usado para criar diferentes tipos de detalhes.

- QRLabel



Textos estático no relatório são formados pelo componente QRLabel bastando para isso modificar a propriedade **Caption**. É possível também modificar a propriedade Caption durante a geração do relatório

- QRMemo



O componente QRMemo e usado para imprimir multiplas linhas de um campo.

- QRDBText



Este componente é responsável pela impressão dos campos dos arquivos.

- QRDBCalc



Componente utilizado para realizar cálculos básicos durante a geração do relatório

- QRSysData



Imprime várias informações sobre o sistema tais como: número da página, data, hora ou título do relatório.

- QRShape




Utilizado para desenhar simples figuras geométricas.

- QRPreview



Este objeto é responsável pela modificação no Preview padrão do relatório.

Vamos agora realizar um exemplo simples e prático com o Quick, criando o mesmo relatório proposto com o **Report Smith**, deste modo acredito, que você pode comparar a facilidade de ambos os geradores.

1. Inicialmente Clique no botão  (New Form) na *Speed Bar*, ou no menu principal a Clique no menu principal a opção **File** e **New...**, em *New Items*, na página *New* e clique no objeto entitulado *Form* e altere a propriedade *name* para **F_Relato** e salve o formulário como **fRelato**.
2. Coloque os seguintes objetos e faça as seguintes alterações:

Query (Localizado na página **Data Access**) , altere as seguintes propriedades:

Propriedade	Valor	Descrição
DatabaseName	BaseDisco	Nome do Alias ou a localização do diretório das tabelas
SQL	Select B.Cod_Disco, B.Nom_Disco, B.Tip_Disco, C.Des_Categ from Basico as B left join Categor as C on (B.Sig_Categ = C.Sig_Categ) where B.Cod_Disco Between :Cod01 and :Cod02 order by B.Cod_Disco	Cláusula de consulta
Params	Acerte ambos os campos para Data Type como AsFloat	Parâmetro da consulta
Name	QryBasico	Nome do objeto

DataSource (Localizado na página **Data Access**) , altere as seguintes propriedades:

Propriedade	Valor	Descrição
DataSet	QryBasico	Nome da tabela vinculada
Name	DSBasico	Nome do objeto

- **QuickReport** (Localizado na página **QReport**)

Propriedade	Valor	Descrição
DataSource	DsBasico	Nome do DataSource vinculado
Name	QrConfere	Nome do objeto
ReportTitle	Cadastro de CD's	Título do Relatório

- Para o Objeto **QrBand** (Localizado na página **QReport**)

Propriedade	Valor	Descrição
BandType	rbPageHeader	Tipo da banda (Cabeçalho de Página)
Name	bdCabeçalho	Nome do objeto
Color	clNavy	Cor da Banda

- Para o Objeto **QrSysData** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdCabeçalho**

Propriedade	Valor	Descrição
AlignToBand	True	Alinha considerando a banda
AutoSize	True	Dimensiona automaticamente o tamanho
Data	qrsReportTitle	Tipo do dado a ser mostrado
Font	Arial, 14, Negrito, Branco	Fonte do objeto
Alignment	taCenter	Alinhamento do objeto

- Para o Objeto **QrSysData** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdCabecalho**

Propriedade	Valor	Descrição
AlignToBand	True	Alinha considerando a banda
AutoSize	True	Dimensiona automaticamente o tamanho
Data	qrsDateTime	Tipo do dado a ser mostrado
Font	Arial, 8, Normal, Branco	Fonte do objeto
Alignment	taRightJustify	Alinhamento do objeto

- Para o Objeto **QrBand** (Localizado na página **QReport**)

Propriedade	Valor	Descrição
BandType	rbColumnHeader	Tipo da banda (Cabeçalho de Coluna)
Name	bdCabColuna	Nome do objeto
Color	clNavy	Cor da Banda
LinkBand	bdCabecalho	Banda de dependência

- Para o Objeto **QrLabel** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdCabColuna**

Propriedade	Valor	Descrição
Caption	Código	Label do objeto
Font	Arial, 12, Negrito, Branco	Fonte do objeto

- Para o Objeto **QrLabel** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdCabColuna**

Propriedade	Valor	Descrição
Caption	Nome	Label do objeto
Font	Arial, 12, Negrito, Branco	Fonte do objeto

- Para o Objeto **QrLabel** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdCabColuna**

Propriedade	Valor	Descrição
Caption	Tipo	Label do objeto
Font	Arial, 12, Negrito, Branco	Fonte do objeto

- Para o Objeto **QrLabel** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdCabColuna**

Propriedade	Valor	Descrição
Caption	Categoria	Label do objeto
Font	Arial, 12, Negrito, Branco	Fonte do objeto

- Para o Objeto **QrBand** (Localizado na página **QReport**)

Propriedade	Valor	Descrição
BandType	rbDetail	Tipo da banda (Cabeçalho de Coluna)
Name	bdDetalhe	Nome do objeto
LinkBand	bdCabColuna	Banda de dependência

- Para o Objeto **QrDbText** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdDetalhe**

Propriedade	Valor	Descrição
AutoSize	True	Dimensiona automaticamente o tamanho
DataSource	DsBasico	DataSource vinculado
DataField	Cod_Disco	Campo vinculado

- Para o Objeto **QrDbText** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdDetalhe**

Propriedade	Valor	Descrição
AutoSize	True	Dimensiona automaticamente o tamanho
DataSource	DsBasico	DataSource vinculado
DataField	Nom_Disco	Campo vinculado

- Para o Objeto **QrDbText** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdDetalhe**

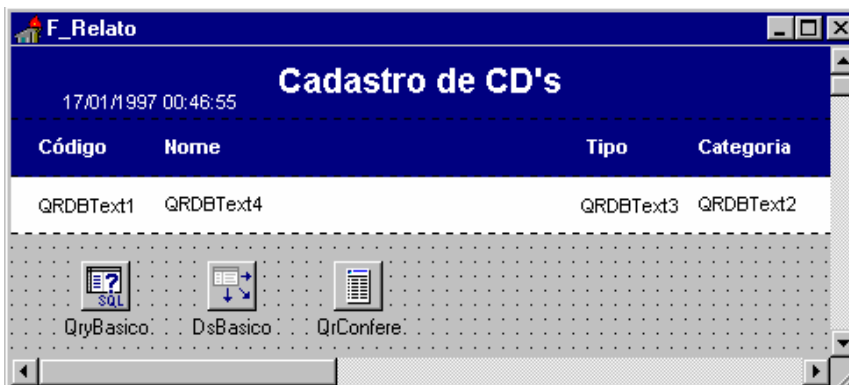
Propriedade	Valor	Descrição
AutoSize	True	Dimensiona automaticamente o tamanho
DataSource	DsBasico	DataSource vinculado
DataField	Tip_Disco	Campo vinculado

- Para o Objeto **QrDbText** (Localizado na página **QReport**), crie-o clicando dentro do objeto **bdDetalhe**

Propriedade	Valor	Descrição
AutoSize	True	Dimensiona automaticamente o tamanho
DataSource	DsBasico	DataSource vinculado
DataField	Des_Categ	Campo vinculado

Criaremos um relatório simples com cabeçalho (BdCabeçalho), Impressão das linhas detalhe (bdDetalhe) e rodapé (bdRodape), a ordem de disposição das bandas não importa, o objeto QuickReport é responsável por este controle.

Ao final acerte a tela de modo que fique semelhante a esta:



3. Por mais incrível que isto possa parecer nosso relatório está pronto, precisamos somente mudar a chamada a partir do objeto F_Menu (aproveitaremos o objeto F_DgGeral criado para o uso com o **Report Smith**) chame novamente o objeto **F_Menu** e clique na opção "Relatório | Geral" chamando o evento onClick;

4. Insira o comando

```
procedure TF_Menu.Geral1Click(Sender: TObject);
begin
  if F_DgGeral.ShowModal = mrOK then           Chama e verifica se a DgGeral retornou OK
  with F_Relato do                             Para o objeto F_Relato
  begin
    QryBasico.Params[0].AsString := StrToFloat(F_DgGeral.CampInicial);  Envia os parâmetros
    QryBasico.Params[0].AsString := StrToFloat(F_DgGeral.CampFinal);
    QryBasico.Open;                      Abre a Query
    QrConfere.Preview;                  Chama o relatório em tela
    QryBasico.Close;                    Fecha a Query
  end;
end;
```

Poderíamos ficar criando “n” tipos de relatórios diferentes mas o melhor método é que você dê uma olhada no diretório [Diretório de Instalação do *Delphi*]\Demos\QuickRpt e execute o projeto **Qrdemo.dpr**. Qualquer outra referência pode ser encontrada no documento **Word** que acompanha o produto, o arquivo **QrManual.doc**.

U' O usuário do *Delphi 1.0* pode ficar se perguntando porque adquirir ou aprender a utilizar um outro gerador de relatório quando o *Delphi* já traz gratuitamente o **ReportSmith**? O problema que acontece se restringe a distribuição de um sistema em *Delphi* que utilize relatórios gerados com o **ReportSmith** este exige um run-time para executar (distribuído gratuitamente tanto na versão 1.0 quanto na versão 2.0) ocupando mais espaço na máquina cliente.

U' Necessariamente, não será preciso enquanto você estiver na fase de “desenhando” o formulário executar o sistema para ver como ficou, simplesmente dê um duplo clique no objeto e o relatório será automaticamente gerado. **Obs.** Cuidado que os eventos criados do formulário não serão executados.

Capítulo IX

Multimídia

Este capítulo foi inserido apenas para sanar quaisquer dúvidas existentes quanto ao desenvolvimento de aplicações que envolvam multimídia com o *Delphi*, a primeira parte não faz parte do desenvolvimento do projeto piloto iniciado.

O que é multimídia ?


Multimídia é uma associação que decorre com o uso de imagens, sons e movimentos, os três tipos de arquivos que se utilizam deste formato de aplicação são:

1. tipo AVI - inclui as produções de vídeo.

2. tipo MID - arquivos para a produção de música utilizando a interface de Instrumentos Musicais Digitalizados, ou formato MIDI.
3. tipo WAV - mais comuns, inclui o registro de sons utilizando a tecnologia Microsoft's WAVE..


O problema principal que ocorre quanto a aplicações multimídia é o espaço físico ocupado, por exemplo, arquivos do tipo AVI, comparando um filme de apenas um minuto ou menos pode ocupar cerca de 5Mb ou até mesmo 10Mb de espaço em disco.

Delphi and Multimedia

O objeto TMediaPlayer , encontrado na *Component Pallete* na página *System*, permite o acesso aos arquivos multimídia. O controle é extremamente simples. De fato, é fácil criar aplicações que envolvam som ou imagens em movimento com apenas uma ou duas linhas de código.

Através deste objeto você tem acesso e controle a rotinas mais internas através da MCI (*Media Control Interface*). Essas rotinas podem ser programadas para acessar toda a possibilidade do mundo multimídia. Com este objetos estas rotinas se tornam extremamente simples e intuitivas para seu uso, como veremos a seguir.

Objeto TMediaPlayer

Para criar uma simples aplicação multimídia crie um novo projeto, e arraste o objeto  para o formulário, automaticamente é criado uma barra de tarefas multimídia, conforme a figura abaixo:



Clique no objeto e altere a propriedade *FileName* para C:\WINDOWS\CHIMES.WAV esta propriedade associa a arquivos tipo AVI, MIDI ou WAVE altere também a propriedade *AutoOpen* para *True*, esta propriedade inicia automaticamente o arquivo.

Depois de completos estes simples passos você já pode rodar o programa. Pressione o botão verde para ouvir o som do arquivo selecionado. Mas de repente você não ouviu nada, não se desespere a causa pode ser um destes problemas


1. Você entrou com o nome do arquivo inválido.
2. Seu sistema de multimídia não está correto.
3. A propriedade *AutoOpen* não está *true*.

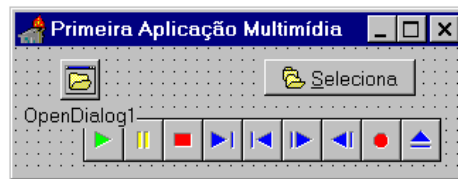
U' Resultados idênticos ocorrem com arquivos MIDI ou AVI.

Colocando as propriedades em modo Runtime

Dependendo das circunstâncias é preferível que o arquivo não esteja sempre aberto mas apenas quando o usuário clicar em um determinado botão. Isto pode ser obtido facilmente modificando a propriedade *AutoOpen* para *False* e no evento *OnClick* do botão insira o seguinte comando:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    MediaPlayer1.Open;
end;
```

Podemos também alterar o arquivo a qual queremos ouvir, para isto insira o objeto **OpenDialog**  encontrado na *Component Palette* na página *Dialogs*, e um objeto *BitBtn* conforme a figura abaixo:



Para o evento *onClick* do objeto **Seleciona** insira o seguinte código:

```
procedure TForm1.SelecionaClick(Sender: TObject);
begin
    MediaPlayer1.Close;
    if OpenDialog1.Execute then
    begin
        MediaPlayer1.FileName := OpenDialog1.FileName;
        MediaPlayer1.Open;
    end;
end;
```

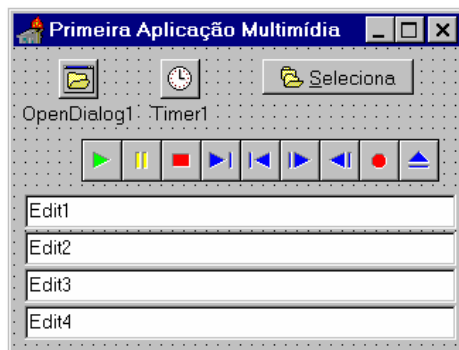
Um ajuste pode ser feito para permitir que o objeto seleção tenha acesso apenas as extensões AVI , WAV, or MID. Podendo ser colocado de duas maneiras diferentes, na propriedade *Filter* do objeto **OpenDialog1**:

1. Arquivos Multimídia (*.avi; *.wav; *.mid) | *.avi;*.wav;*.mid
2. Arquivo AVI (*.avi) | *.avi
Arquivo WAVE (*.wav) | *.wav
Arquivo MIDI (*.MID) | *.mid

U' As barras (|) são utilizadas para dividir duas seções: *Filter Name* e *Filter*.

Pesquisando variáveis em modo RunTime

Uma amostra muito simples de um programa multimídia, e um tanto poderoso que quase qualquer pessoa é capaz de fazê-lo, pois o programa é um pouco robusto, e não aconselho a tentativa por programadores inexperientes, para as informações aqui necessitadas é preciso ter um controle total das capacidades multimídia.



Antes de iniciarmos vamos fazer uma pequena observação, principalmente relativo a dois pontos:

Para alguns programadores que procuram coisas úteis, as informações aqui presentes contém o essencial para transformá-los em programadores multimídia.

Outra observação seria a respeito de alguns controles a arquivos multimídia. Dê uma olhada no arquivo [DiretórioDelphi]\SOURCE\RTL\WIN\MMSYSTEM.PAS, é uma biblioteca de funções que contém todas as chamadas de acesso a baixo nível de comandos Windows para aplicações multimídia. As técnicas de acessos estão contidas no próprio documento.

Com estes dois pontos frescos em nossa mente, podemos iniciar o nosso estudo sobre os aspectos do objeto *TMediaPlayer*.

Dê uma olhada na *Objeto Inspector* na página de Eventos do objeto *TMediaPlayer* você encontrará dois métodos:

O evento *OnClick* que ocorre quando é pressionado qualquer botão do controle. Por instância, através do parâmetro enviado *Button* é possível saber se o botão pressionado foi o *OnPlay*.

Um segundo evento consiste no *OnNotify* por conter a mensagem *mm_MciNotify* que são as chamadas do Windows para o início ou o término de uma execução, com seus eventuais erros.

Ambos os eventos serão discutidos nos próximos parágrafos.

É possível identificar o botão pressionado através do evento *OnClick*, aqui estão todos os tipos gerados pelo *TMPBtmType*:

- ✓ *btPlay*: Quando pressionado o botão verde, Iniciar.
- ✓ *btPause*: Quando pressionado o botão amarelo, Pausa.

- ✓ btStop: Quando pressionado o botão vermelho, Parar.
- ✓ btBack: Quando pressionado o botão azul, Avança a imagem.
- ✓ btStep: Quando pressionado o botão azul, Retorna a imagem.
- ✓ btNext: Quando pressionado o botão azul, Avanço rápido.
- ✓ btPrev: Quando pressionado o botão azul, Retorno rápido.
- ✓ btRecord: Quando pressionado o botão vermelho, Gravação.
- ✓ btEject: Quando pressionado o botão azul, Retirar.

Inicialmente vamos determinar qual foi o tipo de botão pressionado, para tanto, crie para o evento *OnClick* o seguinte código:

```
procedure TForm1.MediaPlayer1Click(Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean);  
begin  
  case Button of  
    btPlay: Edit1.Text := 'Tocando';  
    btPause: Edit1.Text := 'Pausado';  
    btStop: Edit1.Text := 'Parado';  
    btNext: Edit1.Text := 'Próximo';  
    btPrev: Edit1.Text := 'Anterior';  
    btStep: Edit1.Text := 'Avançando';  
    btBack: Edit1.Text := 'Retornando';  
    btRecord: Edit1.Text := 'Gravando';  
    btEject: Edit1.Text := 'Retirando';  
  end;  
end;
```

Para encontrar o que aconteceu com o processo, precisamos do evento *OnNotify*. Aqui estão as mensagens enviadas pelo sistema operacional:

- ✓ mci_Notify_Successful: Comando completado com êxito
- ✓ mci_Notify_Superseded: Comando suspenso por outra função
- ✓ mci_Notify_Aborted: Função corrente foi interrompida
- ✓ mci_Notify_Failure: Algum erro ocorreu.

O *Delphi* não reconhece estas diretivas de mensagem, mas ele converte para constantes do tipo:

- ✓ nvSuccessful indicando o êxito.
- ✓ nvSuperseded indicando que está suspenso, provavelmente por causa de uma pausa.
- ✓ nvAborted messages indicando que foi pressionado o botão parar, ou causa devido ao fechamento do arquivo.

Click no evento *OnNotify* e coloque o seguinte código:

```
procedure TForm1.MediaPlayer1Notify(Sender: TObject);  
var  
  S: String;  
  Total: Integer;  
begin  
  case MediaPlayer1.NotifyValue of  
    nvSuccessful: begin  
      Inc(Total);  
      S := 'mci_Notify_Successful ' + IntToStr(Total);  
    end;
```

```
nvSuperseded: S := 'mci_Notify_Superseded';
nvAborted: S := 'mci_Notify_Aborted';
nvFailure: S := 'mci_Notify_Failure';
else
  S := 'Não consigo identificar a mensagem';
end;
Edit2.Text := S;
if (MediaPlayer1.NotifyValue = nvSuccessful) and (MediaPlayer1.Mode = mpStopped) then
  Edit1.Text := 'Arquivo finalizado';
end;
```

Estes eventos verificam os acontecimentos mais significativos que ocorreram com o dispositivo MCI. O modo corrente com que o dispositivo MCI é especificado também pode ser utilizado pelo objeto *TMediaPlayer*. Aqui uma listagem dos valores mais comuns designados:

- ✓ mci_Mode_Not_Ready
- ✓ mci_Mode_Stop
- ✓ mci_Mode_Play
- ✓ mci_Mode_Record
- ✓ mci_Mode_Seek
- ✓ mci_Mode_Pause
- ✓ mci_Mode_Open

Estes valores são auto-explicativos. Por exemplo, o modo do campo é fixado em *mci_Mode_Stop*, o dispositivo está parado. Se fixado em *mci_Mode_Play*, o dispositivo está tocando.

Crie uma nova procedure *Private* chamada **SetMode**, e insira o seguinte código:

```
procedure TForm1.SetMode;
begin
  Edit4.Text := MediaPlayer1.FileName;
  case MediaPlayer1.Mode of
    mpNotReady: Edit3.Text := 'mci_Mode_Not_Ready';
    mpStopped: Edit3.Text := 'mci_Mode_Stop';
    mpPlaying: Edit3.Text := 'mci_Mode_Play';
    mpRecording: Edit3.Text := 'mci_Mode_Record';
    mpSeeking: Edit3.Text := 'mci_Mode_Seek';
    mpPaused: Edit3.Text := 'mci_Mode_Pause';
    mpOpen: Edit3.Text := 'mci_Mode_Open';
  else
    begin
      Edit1.Text := 'Dispositivo Inativo';
      Edit2.Text := 'Sem mensagens';
      Edit3.Text := 'Não identificado';
      Edit4.Text := 'Não há arquivo selecionado';
    end;
  end;
end;
```

Para a chamada desta rotina click no objeto *Timer* e chame o evento *OnTimer* e insira o seguinte código:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

```
SetMode;  
end;
```

Com a propriedade *Interval* deste objeto fixada em 1000, significa que a cada 1000 milissegundos será disparada a rotina de verificação que informará o estado para o objeto *TMediaPlayer*.

Execute o projeto e atente para os seguintes detalhes:

- Toda a vez que for usado um botão da barra multimídia, será disparado o evento `onClick` marcando deste modo o tipo de botão pressionado;
- As mensagens da `mm_MciNotify` ocorrem durante toda a execução da aplicação, tente utilizar os botões Pausa e Parar no meio de uma execução.
- Quando for selecionar um novo arquivo, afaste um pouco a janela de diálogo e observe como estão os campos *edit*.

U' Observe e estude também o arquivo [DiretórioDelphi]\SOURCE\VCL\MPLAYER.PAS ele é a unidade principal de criação do objeto *TMediaPlayer*.

Inserindo o multimídia para o Sistema


Que tal se além de cadastrarmos os nossos CD's pudéssemos ouvi-los, adicione um opção do menu principal do sistema que chamará o seguinte formulário.

Desenvolvimento do CD Player

- Inicialmente, crie um novo objeto **Form** baseado na template **Blank form** e altere as seguintes propriedades:

Propriedade	Valor	Descrição
BorderStyle	bsDialog	Estilo da borda do formulário
Caption	CD Player	Label do objeto (Tarja azul do formulário)
Name	F_Player	Nome do objeto
Position	poScreenCenter	Posição da janela (centralizado)


- Crie para esta nova janela os seguintes objetos:

TMediaPlayer , encontrado na *Component Pallete* na página *System*, e altere as seguintes propriedades:

Propriedade	Valor	Descrição
VisibleButtons	[btPlay,btPause,btStop,btNext,btPrev,btEject]	Botões que ficarão visíveis
Name	CD	Nome do objeto

 TTimer, encontrado na *Component Pallete* na página *System*, servirá para controlar o tempo das músicas.

 Panel, encontrado na *Component Pallete* na página *Standard*, e altere as seguintes propriedades:

Propriedade	Valor	Descrição
Align	alBottom	Alinhamento dentro do <i>form</i> , todo no rodapé
Alignment	taLeftJustify	Alinhamento da Caption , justificado à esquerda
BevelInner	bvLowered	Borda 3D interna, tipo pressionado
BevelOuter	bvLowered	Borda 3D externa, tipo pressionado
BorderWidth	1	Tamanho da borda
Caption	Insira o CD	Label do objeto
Name	LinhaStatus	Nome do objeto
Font	MS Sans Serif, Estilo da fonte: Normal, Tamanho: 8, Cor: Azul Marinho	Tipo de letra a ser mostrada no objeto, para alterar esta propriedade clique no botão 
Height	22	Altura do objeto

 Panel, encontrado na *Component Pallete* na página *Standard*, e altere as seguintes propriedades:

Propriedade	Valor	Descrição
Align	alTop	Alinhamento dentro do <i>form</i> , todo no topo
BevelInner	bvLowered	Borda 3D interna, tipo pressionado
BevelOuter	bvRaise	Borda 3D externa, tipo pressionado
BorderWidth	2	Tamanho da borda
Height	50	Altura do objeto
Caption		Label do objeto

Crie quatro objetos **Label** dentro do objeto Panel2, para os dois primeiros altere a propriedade **Caption** para **Trilha:** e **Posição:** respectivamente para os outros dois altere a propriedade **Name** para **LblTrack** e **LblTime**, altere a propriedade **Font** de todos para MS Sans Serif, Estilo da fonte: Normal, Tamanho: 8, Cor: Castanho.

Compare o desenho do formulário a seguir:



Quanto ao programa vou explicá-lo na íntegra, acompanhe a listagem colocando os procedimentos nos locais indicados:

unit Fplayer;

interface

uses

SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, ExtCtrls, MPlayer, MMSystem;

type

TF_Player = **class**(TForm)

CD: TMediaPlayer;

Timer1: TTimer;

LinhaStatus: TPanel;

Panel2: TPanel;

Label1: TLabel;

Label2: TLabel;

LblTrack: TLabel;

LblTime: TLabel;

procedure Timer1Timer(Sender: TObject);

procedure CDPostClick(Sender: TObject; Button: TMPBtnType);

procedure CDNotify(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

TrilhaCorrente: byte;

FinalTrilhaCorrente: Longint;

TrilhaLidas: boolean;

CDPlaying, CDPaused: boolean;

TamTrilha: **array**[1..100] **of** LongInt;

function CDPos(Sender: TObject; Trilha, Min, Sec: byte): Longint;

procedure InitCD(Sender: TObject);

procedure ResetCD(Sender: TObject);

public

{ Public declarations }

end;

var

F_Player: TF_Player;

implementation

{ \$R *.DFM }

```
const
  ModeStr: array[TmpModes] of string[10] = ('Não Lido', 'Parado', 'Tocando',
    'Gravando', 'Pesquisando', 'Pausado', 'Aberto');

{ Criado - Calcula a posição }
function TF_Player.CDPos(Sender:TObject; Trilha, Min, Sec: byte): Longint;
var
  i: integer;
begin
  result := 0;
  for i := 1 to (Trilha - 1) do
    begin
      Inc(Result, mci_MSF_Second(TamTrilha[i]));
      Inc(Result, mci_MSF_Minute(TamTrilha[i]) * 60);
    end;
  Inc(Result, Sec);
  Inc(Result, Min * 60);
end;

{ Criado - Inicializa os drivers de CD }
procedure TF_Player.InitCD(Sender:TObject);
var
  i : integer;
begin
  if not TrilhaLidas then
    begin
      LinhaStatus.Caption := 'Lendo as trilhas';
      LinhaStatus.Update;
      for i := 1 to CD.Tracks do
        TamTrilha[i] := CD.TrackLength[i];
      TrilhaLidas := True;
    end;
end;

{ Criado - Reseta o formulário }
procedure TF_Player.ResetCD(Sender:TObject);
begin
  { Limpa para trilha 0 }
  TrilhaCorrente := 1;
  LblTrack.Caption := '0';
  LblTime.Caption := '00:00';

  { Modifica a cor para vermelho }
  LblTrack.Font.Color := clRed;
  LblTime.Font.Color := clRed;

  { Para o CD Player }
  CD.Stop;

  { Coloca a posicao da trilha para a inicial }
  CD.StartPos := mci_Make_TMSF(1, 0, 0, 0);

  CDPlaying := False;
  CDPaused := False;
end;
```

```
{ Evento Timer do Objeto Timer1 }
procedure TF_Player.Timer1Timer(Sender: TObject);
var
  Trilha, Minutes, Seconds : byte;
  strMinuto, strSegundo: String;
  Pos: LongInt;
begin
  { Acerta o estado da linha }
  if CDPaused then
    LinhaStatus.Caption := 'Pausado'
  else
    LinhaStatus.Caption := ModeStr[CD.Mode];

  case CD.Mode of
    mpStopped:
      begin
        { Inicializa o CD para ser lido }
        if not TrilhaLidas then
          InitCD(CD);

        { No caso de ser Continuous Play }
        if CDPlaying and (not CDPaused) then
          begin
            CD.StartPos := mci_Make_TMSF(1, 0, 0, 0);
            CD.Play;
            CD.EnabledButtons := [btPause, btStop, btNext, btPrev, btEject];
          end;
        end;
    mpOpen:
      begin
        TrilhaLidas := False;
        ResetCD(CD);
      end;
    mpPlaying:
      begin
        Pos := CD.Position;
        Trilha := mci_TMSF_Track(Pos);
        Minutes := mci_TMSF_Minute(Pos);
        Seconds := mci_TMSF_Second(Pos);
        TrilhaCorrente := Trilha;

        { Constrói os minutos e segundos para mostrar }
        strMinuto := IntToStr(Minutes);
        strSegundo := IntToStr(Seconds);
        if Length(strMinuto) < 2 then
          strMinuto := '0' + strMinuto;
        if Length(strSegundo) < 2 then
          strSegundo := '0' + strSegundo;

        { Mostra os Labels }
        LblTrack.Caption := IntToStr(Trilha);
        LblTime.Caption := strMinuto + ':' + strSegundo;
      end;
    end;
  end;
```

{ Evento PostClick do Objeto CD }

procedure TF_Player.CDPostClick(Sender: TObject; Button: TMPBtnType);

begin

case Button of

btPlay:

begin

{ Modifica a cor para Azul Marinho }

LblTrack.Font.Color := clNavy;

LblTime.Font.Color := clNavy;

CDPlaying := True;

CDPaused := False;

Update;

end;

btPause:

begin

{ Modifica a cor para Roxo }

LblTrack.Font.Color := clPurple;

LblTime.Font.Color := clPurple;

CDPlaying := False;

CDPaused := True;

Update;

end;

btStop:

ResetCD(CD);

btEject:

begin

TrilhaLidas := False;

ResetCD(CD);

end;

end;

Timer1Timer(Timer1);

end;

{ Evento OnNotify do Objeto CD }

procedure TF_Player.CDNotify(Sender: TObject);

begin

Timer1Timer(Timer1);

end;

{ Evento OnCreate do Objeto F_Player }

procedure TF_Player.FormCreate(Sender: TObject);

begin

CDPlaying := False;

CDPaused := False;

TrilhaLidas := False;

end;

end.

Salve o formulário com o nome de **FPlayer** e faça a chamada a partir do menu principal do seu sistema, bom divertimento.

Capítulo X

Novos Componentes

Uma das maiores vantagens do *Delphi* sobre os demais concorrentes é o fato da geração de novos componentes (de novos objetos).

O exemplo a seguir pretende colocar uma luz sobre o assunto mostrando os passos básicos para o desenvolvimento de componentes, estes passos são:

- Criando propriedades e métodos;
- Controle ao acesso as propriedades;
- Propriedades de leitura e escrita;
- Enviando e recebendo mensagens através dos componentes.

Criando Componentes

Componentes são como blocos de construção para as aplicações *Delphi*. Você poderá construir uma aplicação simplesmente adicionando estes blocos e modificando os eventos, propriedades ou métodos. Todos os componentes possuem duas propriedades em comum: *Name* e *Tag*. Alguns componentes estão distribuídos na *Component Palette*. Mas alguns componentes (*TApplication*, *TMenu*, *TMenuItem*, e *TScreen*) são disponíveis apenas através de seu código.

Você pode criar novos componentes utilizando os seguintes passos:

1. Derivando os novos componentes de um componente já existente.
2. Modificando um componente.
3. Registrando um componente.

O componente é criado como uma *unit* separada de um projeto, podendo ser formado por uma ou mais *unit*'s. Após você criação do componente, compilação e instalação dentro da paleta de componentes. Para usar o componente, selecione-o da *Component Palette* e adicione-o ao formulário.

A Classe *TComponent*

Você criará um novo componente utilizando diretamente o *Code Editor*, para isso você usará a *Component Expert*. Na verdade todos os componentes criados serão derivados de componentes já existentes, mesmo que você deseje criar um componente sem eventos ou propriedades ele será herdado de uma classe já existente a *TComponent*.

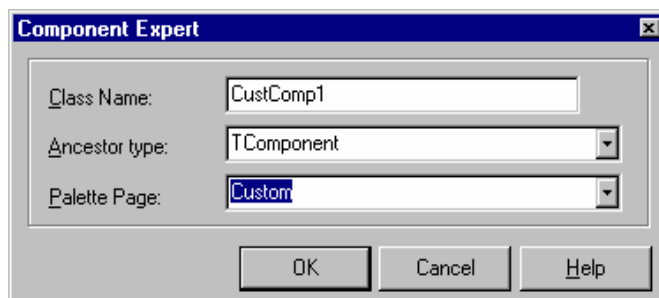
A *TComponent* é uma classe inicial de componentes, sob ela é que foi feita a árvore de componentes *Delphi*, por exemplo, a classe *TControl*, possuem mais de 70 componentes descendentes, tais como: *TBitBtn*, *TButton*, *TCheckBox*, *TColorDialog*, *TComboBox*, *TForm*, *TFontDialog*, *TGroupBox*, *THeader*, *TImage*, *TLabel*, *TListBox*, *TMainMenu* e *TMediaPlayer*. E você ainda pode derivar mais alguns descendentes daqui.

Um Componente Simples

Vamos criar agora um novo componente, para tanto abra um novo projeto e selecione



File | New... selecione a página **New** e o item **Component**. Será mostrada a janela da *Component Expert*. Informe os seguintes parâmetros conforme o desenho abaixo:



Clique no botão OK para aceitar a entrada. A *Component Expert* criará automaticamente o seguinte código para a chamada da Unit1:

```
unit Unit1;
```

```
interface
```

```
uses
```

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
```

```
type
```

```
  CustComp1 = class(TComponent)
```

```
  private
```

```
    { Private declarations }
```

```
  protected
```

```
    { Protected declarations }
```

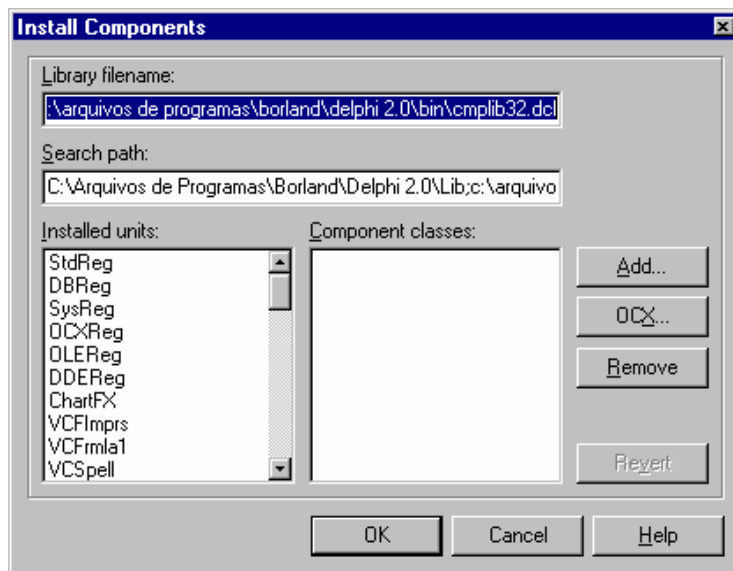
```
  public
```

```
{ Public declarations }  
published  
  { Published declarations }  
end;  
  
procedure Register;  
  
implementation  
  
procedure Register;  
begin  
  RegisterComponents('Custom', [CustComp1]);  
end;  
  
end.
```

Adicionando o Componente a Palheta

Por incrível que pareça mas você terminou de construir seu primeiro componente, e bem verdade que ele não faz absolutamente nada além de abrigar as únicas propriedades padrões existentes em qualquer componente: *Name* e *Tag*. Mas já estamos no começo.

Salve o componente como **Cust**, selecionando **File | Save**, dentro do diretório de instalação do *Delphi* na pasta **Lib**. Para instalar o componente selecione **Component | Install...** aparecerá a seguinte janela diálogo:



Clique no botão **Add...** e será mostrada a janela diálogo **Add Module**. Use o botão **Browse** para localizar o arquivo **Cust.PAS**, note que ele será remetido a *ListBox Installed units*: e caminho do componente (se você salvou-o em outro diretório) será colocado no **Search Path**.

U' Para novos componentes crie um diretório embaixo do *Delphi* intitulado **Lib2**, pois se você sempre colocar cada novo componente no diretório **Lib**, caso você precise copiar apenas os seus componentes, dificilmente os distinguirá dos componentes padrões do *Delphi*.

U' Lembre-se que o **Search Path** é um campo texto limitado em 255 posições então também não adianta para cada componente novo colocar um diretório separado, pois facilmente você estourará o tamanho do campo **Search Path**.

Finalmente clique no botão OK e a **COMPLIB.DCL** (Biblioteca padrão de componentes) será recompilada. Ao término da compilação, note que foi criada uma nova página na *Component Palette* (Custom) e adicionado o novo componente (CustomComp1).

Para testar seu novo componente crie um novo formulário insira o componente *CustomComp1*. Observe através da *Object Inspector* as propriedades do seu novo componente: *Name* e *Tag*.

Criando Propriedades

Para o nosso componente vamos adicionar uma propriedade que armazenará um valor inteiro, para isto chame novamente a **unit** do componente e insira os códigos abaixo da declaração **private**:

```
type
  CustComp1 = class(TComponent)
  private
    fDemoProp: Integer;
```

Agora criaremos uma propriedade aonde o valor será lido e escrito através desta variável para tanto insira os códigos abaixo da declaração **published**:

```
  published
    property DemoProp: Integer read fDemoProp write fDemoProp;
end;
```

U' A seção **Public** (pública) abriga as variáveis, procedimentos ou funções que podem ser lidos e executados por quaisquer outras **units** que utilizem (através cláusula **Uses**) a **unit** em questão já a seção **Published** (Publicado) é utilizada para inserir propriedades ou eventos aos componentes.

Salve o componente e recompile a biblioteca através das opções **Component | Rebuild Library...**, ao término da recompilação, note que para o componente foi criada uma nova propriedade definida como **DemoProp** que armazena o valores inteiros.

Métodos de Acesso

A propriedade criada pode disparar um procedimento ou uma função para executar determinadas ações (por exemplo, colocando um intervalo válido para a variável criada).

Primeiro, escreva a seguinte função para a função **read** do comando **property**:

```
function Cust.GetDemoProp: integer;
```

```
begin
  Result := fDemoProp;
end;
```

Escreva o procedimento para a função **write** do comando **property**:

```
procedure Cust.SetDemoProp(val: integer);
begin
  if val > 99 then
  begin
    DemoProp := fDemoProp;
    raise exception.create('Valor não pode ser maior que 99');
  end
  else
    fDemoProp := val;
end;
```

Declare a função **GetDemoProp** e o procedimento **SetDemoProp** na seção **private**, conforme o exemplo abaixo:

```
private
  fDemoProp: Integer;
  function GetDemoProp: integer;
  procedure SetDemoProp(val: integer);
```

E para a seção **published** troque as propriedades:

```
published
  property DemoProp: Integer read GetDemoProp write SetDemoProp;
```

Salve o componente e recompile a biblioteca através das opções **Component | Rebuild Library...**, note que a propriedade não mais permitirá valores superiores a **99**.

Criando novos tipos

De modo semelhante aos já descritos uma propriedade pode abrigar uma lista de tipos definidos, para tanto na seção **type** defina o conjunto que abrigará os tipos:

```
type
  TDirecao = (drCima, drBaixo, drLado);
```

Na seção **private** crie uma nova variável com base no tipo definido:

```
private
  fDemoProp: Integer;
  fNovaProp: TDirecao;
```

E finalmente na seção **published** defina a propriedade:

```
published
  property DemoProp: Integer read GetDemoProp write SetDemoProp;
  property NovaProp: TDirecao read fNovaProp write fNovaProp;
```

Salve o componente e recompile a biblioteca através das opções **Component | Rebuild Library...**, teste a nova propriedade.

Pensando em Objetos

Mas para que devemos criar novos componentes? Para aliviarmos os “futuros” trabalhos. Lembra-se quando você copiava aqueles pequenos pedaços de rotinas (do tipo: cálculo de CPF/CGC, cálculo de fatorial, um cabeçalho de relatório...) os objetos servem exatamente para guardarmos estes “pequenos pedaços” de blocos de programação, ou se você preferir o termo serve para **encapsularmos** estes códigos.

Construindo um Objeto

Quando for construir objetos lembre-se que ele deve servir à vários aplicativos, nunca construa um objeto que servirá apenas a um único aplicativo (é perda de tempo).

Todo o sistema (pelo menos **for Windows**) necessita de uma janela **Sobre o Sistema** então vamos transformar a janela sobre criada no **Capítulo IV** em um objeto prático que sirva a qualquer sistema, inicie um novo projeto e crie um novo componente, para tanto abra um novo projeto e selecione **File | New...** selecione a página *New* e o item *Component*. Será mostrada a janela da *Component Expert*. Informe os seguintes parâmetros:

Class Name: SobreDlg

Ancestor type: TComponent

Palette Page: Dialogs

Clique no botão OK para aceitar a entrada. As alterações propostas no objeto estão documentadas com o fonte:

unit SobreDlg;

{ Este objeto permite a criação de uma "caixa sobre" padrão para diversos aplicativos. }

interface

uses

SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
Forms, Dialogs, fSobre;

type

TSobreDlg = class(TComponent)

private

FProductName, FVersion, FCopyright, FComments: string;

public

function Execute: Boolean;

published

property NomeProduto: **string read** FProductName **write** FProductName;

property Versao: **string read** FVersion **write** FVersion;

property Direitos: **string read** FCopyright **write** FCopyright;

property Comentario: **string read** FComments **write** FComments;

end;

```
const
  PROCESSOR_INTEL_386 = 386;
  PROCESSOR_INTEL_486 = 486;
  PROCESSOR_INTEL_PENTIUM = 586;
  PROCESSOR_INTEL_860 = 860;
  PROCESSOR_MIPS_R1000 = 1000;
  PROCESSOR_MIPS_R2000 = 2000;
  PROCESSOR_MIPS_R3000 = 3000;
  PROCESSOR_MIPS_R4000 = 4000;
  PROCESSOR_ALPHA_21064 = 21064;
  PROCESSOR_PPC_601 = 601;
  PROCESSOR_PPC_603 = 603;
  PROCESSOR_PPC_604 = 604;
  PROCESSOR_PPC_620 = 620;

var
  SobreDlg: TSobreDlg;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Dialogs', [TSobreDlg]);
end;

function TSobreDlg.Execute: Boolean;
var
  OsInfo: TOSVERSIONINFO;
  SysInfo: TSYSTEMINFO;
  MemStat: TMEMORYSTATUS;
  DiskNo: Integer;
begin
  // Cria a janela em memória
  F_Sobre := TF_Sobre.Create(Application);

  try
    with F_Sobre do
      begin
        // Coloca as propriedades nas variaveis do formulario
        ProductName.Caption := NomeProduto;
        Version.Caption := Versao;
        Copyright.Caption := Direitos;
        Comments.Caption := Comentario;
        Caption := 'Sobre ' + NomeProduto;

        OsInfo.dwOSVersionInfoSize := sizeof(TOSVERSIONINFO);
        GetVersionEx(OsInfo);

        // Versão do Windows
        case OsInfo.dwPlatformId of
          VER_PLATFORM_WIN32s : WinVersion.Caption := 'Windows 3.1';
          VER_PLATFORM_WIN32_WINDOWS : WinVersion.Caption := 'Windows 95';
          VER_PLATFORM_WIN32_NT : WinVersion.Caption := 'Windows NT';
```

```
end;
DosVersion.Caption := format('%d.%d Ver : %d',
  [OsInfo.dwMajorVersion,OsInfo.dwMinorVersion,LOWORD(OsInfo.dwBuildNumber)]);

// Pega o processador
GetSystemInfo(SysInfo);
case SysInfo.dwProcessorType of
  PROCESSOR_INTEL_386      : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Intel 80386']);
  PROCESSOR_INTEL_486      : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Intel 80486']);
  PROCESSOR_INTEL_PENTIUM  : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Intel Pentium']);
  PROCESSOR_MIPS_R1000      : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'MIPS R1000']);
  PROCESSOR_MIPS_R2000      : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'MIPS R2000']);
  PROCESSOR_MIPS_R3000      : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'MIPS R3000']);
  PROCESSOR_MIPS_R4000      : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'MIPS R4000']);
  PROCESSOR_ALPHA_21064     : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'ALPHA 21064']);
  PROCESSOR_PPC_601         : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Power PC 601']);
  PROCESSOR_PPC_603         : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Power PC 603']);
  PROCESSOR_PPC_604         : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Power PC 604']);
  PROCESSOR_PPC_620         : CPU.Caption :=
    format('%d %s',[SysInfo.dwNumberOfProcessors, 'Power PC 620']);
end;

MemStat.dwLength := sizeof(TMEMORYSTATUS);
GlobalMemoryStatus(MemStat);

FreeMemory.Caption := format('Tot: %d KB   Disp: %d KB',
  [Trunc(MemStat.dwTotalPhys/1024),Trunc(MemStat.dwAvailPhys/1024)]);

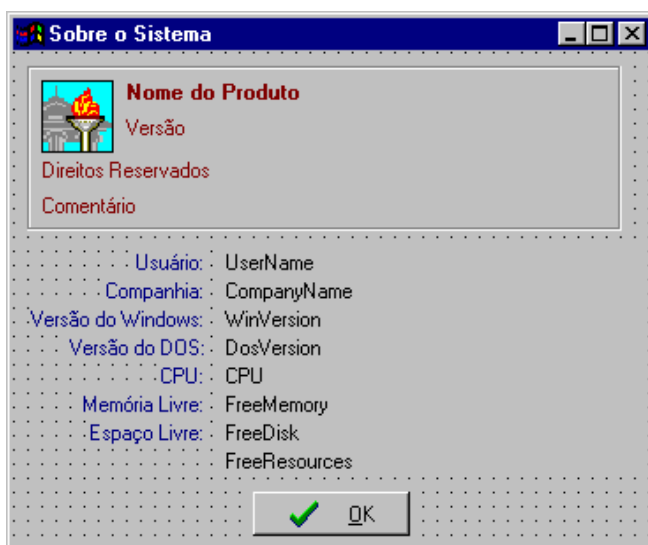
DiskNo := 3;
FreeDisk.Caption := '';
FreeResources.Caption := '';
repeat
  if DiskNo < 7 then
    FreeDisk.Caption := FreeDisk.Caption + format('%s: %d MB ',
      [Chr(DiskNo + Ord('A')- 1),Trunc(DiskFree(DiskNo)/1024/1024)]);
  else
    FreeResources.Caption := FreeResources.Caption + format('%s: %d MB ',
      [Chr(DiskNo + Ord('A')- 1),Trunc(DiskFree(DiskNo)/1024/1024)]);
    inc(DiskNo);
  until DiskFree(DiskNo) = -1;

ProgramIcon.Picture.Graphic := Application.Icon;
Result := (ShowModal = IDOK);
end;
finally
```

```
F_Sobre.Free;
end;
end;

end.
```

Para esta primeira etapa do nosso objeto note que tudo gira em torno do procedimento **Execute** (o nome se deve apenas a uma questão de padronização com os outros objetos da palheta *Dialogs*), através deste procedimento todas as outras variáveis são iniciadas, resta-nos agora a criação do formulário, crie um novo formulário, **ATENÇÃO:** Não aproveite o formulário **F_Sobre** já criado pois este é uma herança do formulário **F_Splash**, crie-o conforme o desenho abaixo:



U' Para os objetos dentro do painel: altere a propriedade **Caption:** Nome do Produto, Versão, Direitos Reservados e Comentário e a propriedade **Name:** ProductName, Version, Copyright, Comments.

U' Para os objetos fora do painel: altere os objetos **Labels** da esquerda a propriedade **Caption** (Ex: Usuário, Companhia) e os da direita a propriedade **Name** (Ex: UserName, CompanyName)

U' A propriedade **Name** para todos os componentes, na ordem que eles aparecem são:

Objeto	Tipo	Objeto	Tipo	Objeto	Tipo	Objeto	Tipo
ProgramIcon	TImage	ProductName	TLabel	Version	TLabel	Copyright	TLabel
Comments	TLabel	Label1	TLabel	UserName	TLabel	Label2	TLabel
CompanyName	TLabel	Label3	TLabel	WinVersion	TLabel	Label4	TLabel
DosVersion	TLabel	Label5	TLabel	CPU	TLabel	Label6	TLabel
FreeMemory	TLabel	Label7	TLabel	FreeDisk	TLabel	FreeResources	TLabel
F_Sobre	TForm	Panel1	TPanel				

Salve a janela com o nome de **FSobre** e o componente como **SobreDlg** e compile a biblioteca e instale o componente: *SobreDlg.PAS*, teste o componente da seguinte forma:

1. Remova do projeto o formulário **F_Sobre** e retire sua chamada da cláusula **Uses**;
2. Coloque o componente no formulário **F_Menu** e acerte as suas propriedades;
3. Insira a chamada ao componente na opção de **Sobre o Sistema**:

```
procedure TF_Menu.ItemAuxilio1Click(Sender: TObject);
```

```
begin
```

```
    SobreDlg.Execute;
```

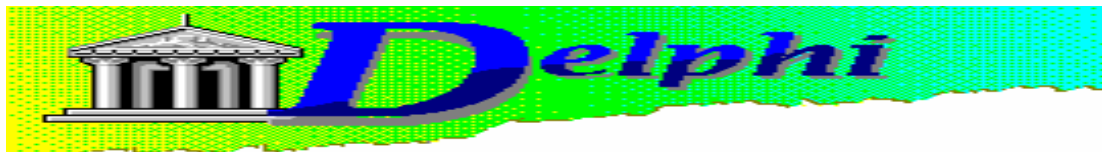
```
end;
```

4. Compile e rode o sistema.

Bem-vindo ao mundo dos objetos.

Finalmente

É facilmente reconhecido que este novo ambiente da *Borland* possui um poder um tanto ilimitado, tanto na criação de sistemas como no desenvolvimento de aplicações de multimídia resta-nos (a nós desenvolvedores) deixarmos a imaginação fluir e iniciar tudo aquilo que sempre desejamos, espero que lhe tenha ajudado ao menos a trilhar o caminho das pedras.



Apêndice A

Documentação

A documentação incluída com o *Delphi Client/Server*:

- *Delphi User's Guide*
- *Delphi Component Writer's Guide*
- *Delphi Database Application Developer's Guide*
- *SQL Links User's Guide*
- *InterBase User's Guide*
- *InterBase Language Reference*
- *InterBase Data Definition*
- *ReportSmith Creating Reports*

Toda a documentação é encontrada em forma de livros e modo on-line (para o segundo caso faz-se necessária a instalação do ACROBAT[®] Reader 2.0 que acompanha o produto).

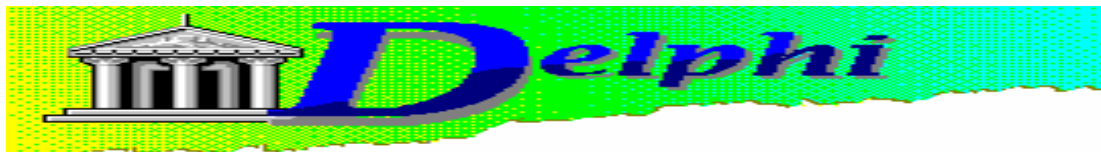
Hardware/Software requeridos

O *Delphi Client/Server* requer:

- *Windows 3.1[®] ou superior*
- *40 megabytes de espaço livre para a instalação mínima*
- *90 megabytes de espaço livre para a instalação completa*
- *um processador 80386 ou maior (486 recomendado)*
- *8 megabytes de RAM*

Para instalar, rode o programa INSTALL.EXE direto do CD ROM ou do disquete e prossiga com as instruções. Serão incluídas informações adicionais no arquivo README.TXT normalmente instalado no diretório \DELPHI.

Alguns exemplos de aplicações estão contidas no diretório \DELPHI\DEMOS.



Apêndice B

Conversão de Campos

É possível, com o *DELPHI*, criarmos um mesmo sistema que rode em diferentes tipos de bases, através de um único **ALIAS**. Para isto precisamos que a definição, tamanho e nome dos campos e tabelas sejam necessariamente os mesmos. Abaixo está a conversão para quatro bases lógicas de dados:

Sintaxe SQL - para *InterBase*, *ORACLE*®, *Informix*® entre outras.

BDE Lógico - A chamada do campo interna ao *DELPHI*.

Paradox - Bases do tipo *Paradox*.

dBASE - Bases do Tipo *.DBF*.

Sintaxe SQL	BDE Lógico	Paradox	dBASE
SMALLINT	fldINT16	Short	Number (6,10)
INTEGER	fldINT32	Long Integer	Number (20,4)
DECIMAL(x,y)	fldBCD	BCD	N/A
NUMERIC(x,y)	fldFLOAT	Number	Number (x,y)
FLOAT(x,y)	fldFLOAT	Number	Float (x,y)
CHARACTER(n)	fldZSTRING	Alpha	Character
VARCHAR(n)	fldZSTRING	Alpha	Character
DATE	fldDATE	Date	Date
BOOLEAN	fldBOOL	Logical	Logical
BLOB(n,1)	fldstMEMO	Memo	Memo
BLOB(n,2)	fldstBINARY	Binary	Binary
BLOB(n,3)	fldstFMTMEMO	Formatted memo	Não Apresenta
BLOB(n,4)	fldstOLEOBJ	OLE	OLE
BLOB(n,5)	fldstGRAPHIC	Graphic	Não Apresenta
TIME	fldTIME	Time	Não Apresenta
TIMESTAMP	fldTIMESTAMP	Timestamp	Não Apresenta
MONEY	fldFLOAT, fldstMONEY	Money	Float (20,4)
AUTOINC	fldINT32, fldstAUTOINC	Autoincrement	Não Apresenta
BYTES(n)	fldBYTES(n)	Bytes	Não Apresenta

x = precisão (default: específico para o driver)

y = escala (default: 0)

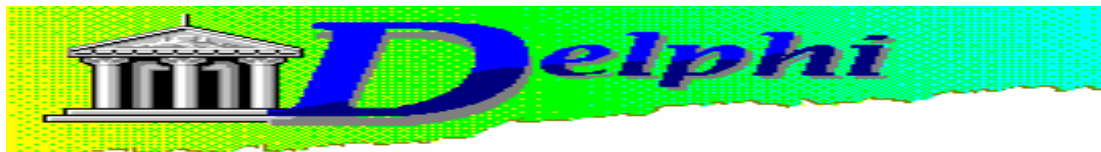
n = tamanho em bytes (default: 0)

1-5 = BLOB subtipo (default: 1)

Tipos de Dados para o InterBase


O InterBase suporta alguns tipos de dados SQL, mas não suporta diretamente dados do tipo **TIME** ou **TIMESTAMP**. A lista abaixo contém os tipos de dados disponíveis para as declarações SQL no InterBase:


Nome	Tamanho	Precisão	Descrição
BLOB	Variável	Não apresenta	Objeto do tipo binário largo, possível guardar dados grandes, tais como gráficos, textos e voz digitalizada.
CHAR(n)	n Caracteres	1 até 32767	Tamanho fixo de caracteres ou tipo de string. O nome também pode apresentar : CHARACTER .
DATE	64 bits	1 jan 100 até 11 jan 5941	Também inclui todas as informações sobre a hora.
DECIMAL (precisão, escala)	variável	Precisão: 1 até 15 e escala de 1 até 15	Especifica-se por precisão o número de dígitos a serem gravados e por escala o número de casas decimais, exemplo DECIMAL(10,3) é igual ao formato: ppppppp.eee.
DOUBLE PRECISION	64 bits	$1,7 \times 10^{-308}$ até $1,7 \times 10^{308}$	Utilizado para valores científicos, com 15 dígitos de precisão.
FLOAT	32 bits	$3,4 \times 10^{-38}$ até $3,4 \times 10^{38}$	Precisão simples, com 7 dígitos de precisão.
INTEGER	32 bits	-2.147.483.648 até 2.147.483.648	Campo do tipo longo.
NUMERIC (precisão, escala)	variável	Precisão: 1 até 15 e escala de 1 até 15	Especifica-se por precisão o número de dígitos a serem gravados e por escala o número de casas decimais, exemplo NUMERIC(10,3) é igual ao formato: ppppppp.eee.
SMALLINT	16 bits	-32.768 até 32.767	Campo médio.
VARCHAR(n)	n Caracteres	1 até 32767	Tamanho variável de caracteres ou tipo de string. O nome também pode apresentar : VARYING CHAR ou VARYING CHARACTER .



Apêndice C

Aplicação rápida com o Objeto Query

Para não ficar quaisquer dúvidas sobre o objeto tQuery , vamos utilizá-lo para a demonstração de uma pequena aplicação, neste exemplo, utilizaremos a base de dados encontrada no diretório **C:\DELPHI\DEMOS\DATA** utilizaremos as tabelas: *Customer*, *Orders*, *Parts* e *Items*. A idéia é mostrar um formulário com o nome do cliente ligado a um **Grid** com todas as encomendas pertencentes a este cliente. Conforme a figura abaixo:

Para começar, coloque em um novo formulário um objeto **tTable**  e um objeto **tQuery**. O objeto **tTable** efetuará a ligação com a tabela de Clientes (Tabela *Customer*), enquanto que o **tQuery** irá extrair os detalhes da encomenda (Tabelas *Orders*, *Parts* e *Items*) apropriados a cada cliente. Os dados estão em uma tabela *Paradox*, pelo que na propriedade **DataBaseName** é colocado o nome do diretório que contém os dados (ou defina o nome do Alias: *DBDemos*). A ligação é completada através da definição das propriedades **TableName** e **IndexName**, e se alternarmos a propriedade **Active** para *true* vemos os dados reais, mesmo durante a fase de construção do formulário.

Se os registros contendo os detalhes dos pedidos estivessem todos em uma única tabela, a junção poderia ser facilmente realizada pela definição das propriedades **MasterSource** e

MasterField de um segundo objeto tabela, mas, uma vez que necessitamos de dados de mais de uma tabela (*Orders* e *Products*), esta técnica simples não funciona. O controle de consulta recupera um conjunto diferente de registros para cada cliente, extraíndo dados das duas tabelas ligadas. O *Delphi* oferece duas técnicas para fazer isto: Uma delas envolve o uso de variáveis Calculadas para uma das três tabelas em questão; uma outra (mais simples) envolve à atribuição de um valor à propriedade **SQL** do objeto **tQuery**, ou seja, uma instrução **SQL** apropriada. Neste exemplo, vamos fazer colocando algum código no evento **OnDataChange** do objeto **DataSource**. Lembre-se que este evento é chamado sempre que o registro corrente é alterado.

Insira a seguinte instrução na propriedade **SQL** do objeto **tQuery**:

```
SELECT Orders.OrderNo, Items.Qty, Parts.Description, Parts.ListPrice
FROM Orders, Items, Parts
WHERE Orders.OrderNo = Items.OrderNo AND
      Items.PartNo = Parts.PartNo AND
      Orders.CustNo = :CustNo;
```

Repare no código **SQL** na variável *:CustNo*, vá para a propriedade **Params** do objeto **tQuery** e coloque para a variável *CustNo* criada o **Data type** como *Float*, esta variável será passada para o **SQL** através do evento **OnDataChange** do objeto **DataSource**, insira o seguinte código:

```
procedure TForm1.DataSource2DataChange(Sender: TObject; Field: TField);
begin
  Query1.Close;
  Query1.Params[0].AsFloat := Table1CustNo.Value;
  Query1.Open;
end;
```

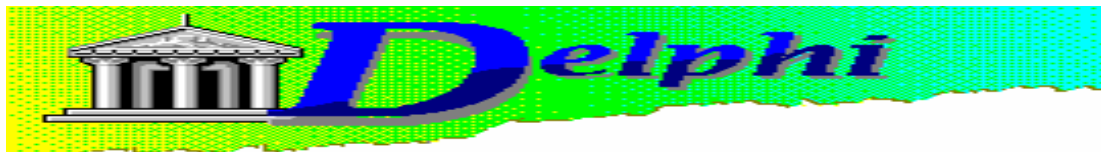
U' Relembrando no *Delphi* os dados fluem da base de dados para o formulário na seguinte sequência:

Base de Dados \Rightarrow objeto **DataSet** (tTable ou tQuery) \Rightarrow objeto **DataSource** \Rightarrow objeto Campo

O objeto **DataSource** é necessário, pois os objetos de campo não podem ligar-se diretamente aos objetos do **DataSet**, mas apenas através de um objeto **DataSource**.

Por conseguinte, para cada objeto **DataSet** também é colocado no formulário um objeto **DataSource**. Finalmente, são adicionados objetos de caixa de edição de texto, um objeto **Grid** para as linhas da encomenda e um objeto **dbNavigator** ligado a tabela de *Customer*, rode o projeto.

- objeto **Grid** está ligado ao objeto **tQuery** (através do objeto **DataSource**), então este se atualiza automaticamente com os novos resultados das consultas.



Apêndice D

Imprimindo um Formulário

Para imprimir um formulário não tem nenhum segredo, existe o comando *PRINT* relacionado a formulários, o problema se inicia quando o formulário ultrapassa as dimensões da tela do seu monitor, ou seja, a largura e altura dele é maior que a tela. Digamos um formulário qualquer que tenha a propriedade `HorzScrollBar.Range = 768` e `VertScrollBar.Range = 1008` (isto corresponde a uma folha de papel tamanho A4).

O programa abaixo resolve exatamente este problema, imprimindo somente objetos: **Tlabel**, **TEdit**, **TMemo**, **TDBText**, **TDBEdit** e **TDBMemo**, utiliza a biblioteca *Printers* para fazer o serviço, coloque um botão qualquer no formulário que deseje imprimir e para o evento `onClick`, digite os seguintes comandos:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);  
var  
    C : array[0..255] of char;  
    CLen, ScaleX, ScaleY, Ind : Integer;  
    Format : Word;  
    DC : HDC;  
    MComp : Tmemo;  
    R : TRect;  
begin  
    Printer.BeginDoc;  
    DC := Printer.Canvas.Handle;  
    ScaleX := GetDeviceCaps(DC, LOGPIXELSX) div PixelsPerInch;  
    ScaleY := GetDeviceCaps(DC, LOGPIXELSY) div PixelsPerInch;  
    for Ind := 0 to ComponentCount - 1 do  
        if (Components[Ind] is TCustomLabel) or (Components[Ind] is TCustomEdit) then  
            begin  
                MComp := TMemor(Components[Ind]);  
                if (MComp.visible) then  
                    begin  
                        Printer.Canvas.Font := MComp.Font;  
                        DC := Printer.Canvas.Handle;  
                        R := MComp.BoundsRect;  
                        R.Top := (R.Top + VertScrollBar.Position) * ScaleY;  
                        R.Left := (R.Left + HorzScrollBar.Position) * ScaleX;  
                        R.Bottom := (R.Bottom + VertScrollBar.Position) * ScaleY;  
                        R.Right := (R.Right + HorzScrollBar.Position) * ScaleY;  
                        if (not (Components[Ind] is TCustomLabel)) and (MComp.BorderStyle = bsSingle) then  
                            Printer.Canvas.Rectangle(R.Left, R.Top, R.Right, R.Bottom);  
                        Format := DT_LEFT;  
                        if (Components[Ind] is TEdit) or (Components[Ind] is TCustomMaskEdit) then  
                            Format := Format or DT_SINGLELINE or DT_VCENTER  
                        else
```

```
begin
  if MComp.WordWrap then
    Format := DT_WORDBREAK;
  if MComp.Alignment = taCenter then
    Format := Format or DT_CENTER;
  if MComp.Alignment = taRightJustify then
    Format := Format or DT_RIGHT;
  R.Bottom := R.Bottom + Printer.Canvas.Font.Height + 1;
end;
CLen := MComp.GetTextBuf(C,255);
R.Left := R.Left + ScaleX + ScaleX;
DrawText(DC, C, CLen, R, Format);
end;
end;
Printer.EndDoc;
Close;
end;
```

Se você conhece um pouco de *Pascal 7.0* não acredito que você teve dificuldades em interpretar o programa, se você não conhece aqui vão algumas dicas:

Inicialmente foi declarado uma série de variáveis que serão utilizadas posteriormente. E então iniciado o objeto de impressão *Printer* através do comando:

```
Printer.BeginDoc;
```

O objeto imprime através de uma subclasse conhecida por *Canvas*. Esta classe é que torna possível a criação de toda a interface gráfica do *Delphi* (Este objeto parte de um encapsulamento da Windows HDC). Uma forma simples de se imprimir seria utilizar os seguintes comandos:

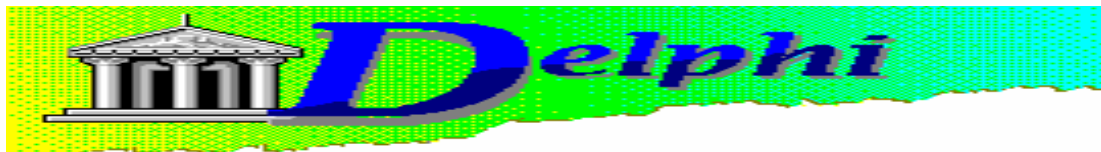
```
Printer.BeginDoc;
Printer.Canvas.TextOut(0, 0, 'Está imprimindo...');
Printer.EndDoc;
```

Mas, para se obter uma cópia fiel do formulário é preciso fazer mais do que isto, inicialmente é chamada a propriedade *handle* (esta faz uma chamada a Windows GDI chamando funções da API para requerer o modo de desenho dos objetos envolvidos), próximo passo é verificar o tipo de escala (em pixels) da largura e altura da janela a ser impressa.

Agora resta verificar objeto a objeto (do tipo *Label* ou do tipo *Edit*) e enviar suas características para os dados da classe *Canvas* e imprimí-los. Lembre-se o padrão de impressão do Windows[®] é emitir a listagem somente quando a mesma estiver completa e isto só acontecerá no comando :

```
Printer.EndDoc;
```

U' Uma outra saída para a impressão de seus relatórios pode ser conseguida através da utilização de inúmeras bibliotecas prontas que fazem o acesso ao objeto *TPrinter*. Entre elas existem a ReportPrint[®] (da **Nevrona Designs**) que pode ser adquirido uma versão de demonstração através da internet através do seguinte endereço: <ftp.primenet.com/users/j/jgunkel/delphi/rprinter.zip>



Apêndice E

Trabalhando com Máscaras

Abriremos aqui este apêndice para esclarecermos a respeito de campos mascarados no *Delphi*, alguns campos podem possuir uma máscara para edição através da propriedade *MaskEdit*, encontrada para os objetos *TDateField*, *TDateTimeField*, *TStringField*, *TTimeField* e o objeto.

Para montar uma máscara, utiliza-se a propriedade **EditMask** observando o limite de dados que o campo poderá armazenar. É utilizada basicamente para a validação ou a formatação da entrada de um determinado campo.

A máscara para os campos pode restringir o uso de determinados caracteres ou formatos válidos, mostrando automaticamente uma janela de não aceitação da máscara. A validação ocorre caracter a caracter. Use o evento *OnValidate* para validar uma entrada completa.

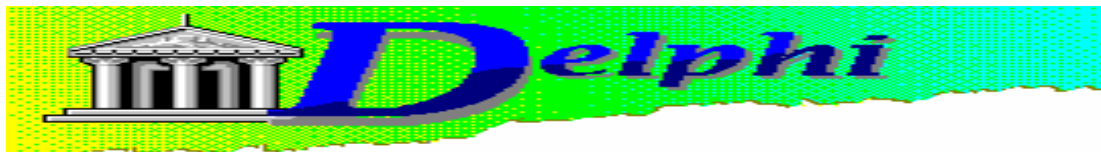
A máscara basicamente consiste de três campos, separados por ponto e vírgula. A primeira parte e a máscara propriamente dita. A segunda parte determina se os caracteres fixos devem ser ou não salvos com a máscara (ex: /, -, (, ...). A terceira parte da máscara representa o caracter em branco, podendo ser substituído por outro (ex: _, @, ...).

Estes são os caracteres especiais utilizados com a máscara:

Caracter	Utilização na máscara
!	Causa a digitação da máscara fique parada no primeiro caracter, fazendo com que os caracteres digitados que se movam. Ex: !0;_
>	Todos os caracteres digitados serão convertidos para maiúsculas. Ex: >aaa;0;_
<	Todos os caracteres digitados serão convertidos para minúsculas. Ex: <aaa;0;_
<>	Anula o uso dos caracteres > e <. Ex: >aaa<>aaa;0;_
\	Utilizado para marcar determinado caracter não especial como fixo. Ex: !(999)000-0000;0;_
L	Requer somente caracteres alfabéticos obrigatórios para a posição, do tipo A-Z, a-z. Ex: LLL;1;_
l	Permite somente caracteres alfabéticos para a posição, mas não-obrigatórios, do tipo A-Z, a-z. Ex: lll;1;_
A	Requer somente caracteres alfanuméricos obrigatórios para a posição, do tipo A-Z, a-z, 0-9. Ex: AAA;1;_
a	Permite somente caracteres alfanuméricos para a posição, mas não-obrigatórios, do tipo A-Z, a-z, 0-9. Ex: aaa;1;_
C	Requer um caracter obrigatório para a posição. Ex: CCC;1;_
c	Permite o uso de qualquer caracter para a posição, limitando apenas o número de caracteres utilizados. Ex: ccc;1;_
0	Requer somente caracteres numéricos obrigatórios para a posição, do tipo 0-9. Ex: 000;1;_
9	Permite somente caracteres numéricos para a posição, não-obrigatórios, do tipo 0-9. Ex: 999;1;_
#	Permite somente caracteres numéricos para a posição e o uso dos sinais de - ou +, não-obrigatórios. Ex: ###;1;_
:	Utilizado como separador de horas, minutos e segundos.
/	Utilizado como separador de dia, mês e ano.

Exemplos de Máscaras:

Tipo	Máscara	Entrada	Formatação	Saída
Telefone	!(999)000-0000;1;_	0613873350	(061)3873350	(061)387-3350
CEP	00000\9999;0;_	73015020	73015-020	73015020
Hora	!90:00:00 >LL;0;_	100043PM	10:00:43 PM	22:00:43



Apêndice F

Trabalhando com Importação e Exportação

A importação e exportação de arquivos no Delphi pode ser realizada sem problemas quando se tratar de tabelas em formatos padrão para o Delphi (Paradox, dBase, Oracle, Sybase, ODBC, etc) o quase problema era se tratando de exportação para arquivos no formato .TXT, quase porque ele será resolvido a partir deste aplicativo.

1. Crie um novo projeto. Grave a unidade padrão como **fExpImp** e o projeto como **ExpImp**.
2. Coloque os seguintes objetos e faça as seguintes alterações:
 - Para o Objeto **Table** (Localizado na página **Data Access**)

Propriedades	Configuração
DataBaseName	DbDemos
TableName	EMPLOYEE

- Para o Objeto **OpenDialog1** (Localizado na página **Dialogs**)

Propriedades	Configuração
DefaultExt	TXT
Filter	Arquivo Texto *.TXT

- Para o Objeto **SaveDialog1** (Localizado na página **Dialogs**)

Propriedades	Configuração
DefaultExt	TXT
Filter	Arquivo Texto *.TXT

- Para o Objeto **Button** (Localizado na página **Standard**)

Propriedades	Configuração
Caption	&Exporta
Name	ButExporta

- Para o Objeto **Button** (Localizado na página **Standard**)

Propriedades	Configuração
Caption	&Importa
Name	ButImporta

O maior trabalho da codificação ficaria por conta de criar janelas de Salvar e Abrir mas todo esse trabalho é realizado pelos objetos **OpenDialog** e **SaveDialog**. O resto do código é bem simples vejamos:

- Para o evento **OnClick** do objeto ButExporta

```
procedure TForm1.ButExportaClick(Sender: TObject);
var
  Arq: TextFile;
begin
  if SaveDialog1.Execute then
  begin
    Screen.Cursor := crHourGlass;
    with Table1 do
    begin
      Open;
      First;
      if not EOF then
      begin
        AssignFile(Arq, SaveDialog1.FileName);
        Rewrite(Arq);
        repeat
          WriteLn(Arq, FieldByName('EmpNo').AsString + '|' +
            FieldByName('LastName').AsString + '|' +
            FieldByName('FirstName').AsString + '|' +
            FieldByName('PhoneExt').AsString + '|' +
            FieldByName('HireDate').AsString + '|' +
            FieldByName('Salary').AsString + '*');
        next;
      until EOF;
      CloseFile(Arq);
    end;
    close;
  end;
  Screen.Cursor := crDefault;
end;
```

Inicialmente vamos exportar o arquivo (no caso Employee), o função do comando **AssignFile** e iniciar um objeto de arquivo texto (determinada pelo tipo de variável **TextFile**) e o comando **Rewrite** prepara o objeto iniciado para a gravação. Lembre-se que um arquivo texto só pode receber logicamente texto definido pelos comandos **Write** (insere um texto em determinado arquivo e o cursor de gravação permanece na posição) e **WriteLn** (insere um texto em determinado arquivo e o cursor de gravação inicia uma nova linha) então o único trabalho será de percorrer o nosso arquivo com o comando **Repeat**.

- Para o evento **OnClick** do objeto ButImporta

```
procedure TForm1.ButImportaClick(Sender: TObject);
var
  Arq: TextFile;
  Texto: String;
  I: Integer;

function MontaVariavel: String;
var
  monta: String;
begin
  monta := '';
  inc(I);
```

```
while Texto[I] <> '*' do
begin
  if Texto[I] = '|' then
    break;
  monta := monta + Texto[I];
  inc(I);
end;
result := monta;
end;

begin
  if OpenFileDialog1.Execute then
  begin
    Screen.Cursor := crHourGlass;
    Table1.Open;
    AssignFile(Arq, OpenFileDialog1.FileName);
    Reset(Arq);
    if not EOF(Arq) then
    repeat
      ReadLn(Arq, Texto);
      with Table1 do
      begin
        Insert;
        i := 0;
        FieldByName('EmpNo').AsString := MontaVariavel;
        FieldByName('LastName').AsString := MontaVariavel;
        FieldByName('FirstName').AsString := MontaVariavel;
        FieldByName('PhoneExt').AsString := MontaVariavel;
        FieldByName('HireDate').AsString := MontaVariavel;
        FieldByName('Salary').AsString := MontaVariavel;
        Post;
      end;
    until EOF(Arq);
    CloseFile(Arq);
    Table1.Close;
    Screen.Cursor := crDefault;
  end;
end;
```

Vamos agora importar o arquivo (no caso Employee), o função do comando **AssignFile** e iniciar um objeto de arquivo texto (determinada pelo tipo de variável **TextFile**) e o comando **Reset** prepara o objeto iniciado para a gravação. Os comandos de leitura são **Read** (Lê um caractere de determinado arquivo) e **ReadLn** (Lê uma linha de determinado arquivo) então o trabalho agora será de separar em pedaços a linha lida isto é realizado na função **MontaVariavel** que lerá pedaços demarcados do arquivo exportado.

Apêndice G

Doze melhores dicas para o Delphi

Neste último apêndice do trabalho reservei uma coisa especial, pesquisei em todos os documentos tipo *Tips & Tricks* (Dicas e Truques) e encontrei um documento que falava sobre as onze melhores dicas para o *Delphi*.

Autor: Paul Harding 100046.2604@Compuserve.Com (algumas foram alteradas)

1. Como fazer para o computador soar o beep:

```
messageBeep(0);
```

2. Como pausar um programa por determinado número de segundos:

```
var
  NumSec: SmallInt;
  StartTime: LongInt;
begin
  StartTime := Now;
  repeat
    Application.ProcessMessages;
  until Now > StartTime + NumSec * (1/24/60/60);
end;
```

3. Como mostrar o mouse como uma ampulheta (e depois retorná-lo ao normal):

```
try
  Screen.Cursor := crHourGlass;
  { Escreva o ação a executar aqui }
finally
  Screen.Cursor := crDefault;
end;
Application.ProcessMessages;
```

4. Como controlar o pressionamento da tecla <Enter>:

```
procedure TForm1.EditKeyPress(Sender: TObject; var Key: Char);
{ através do evento onKeyPress do formulário de controle... }
begin
  { se a "var Key" retornar o código #13 corresponde a <Enter>, #9 corresponde a tecla TAB }
  if Key = #13 then
  begin
    Key := #0 { Suprime o som }
    { escreva aqui os seus comandos }
  end;
end;
```

5. Como modificar a cor do texto dentro de um campo DBGrid dependendo do conteúdo:

```
procedure TForm1.DBGridDrawDataCell(Sender: TObject; const Rect: TRect; Field: TField; State:
TGridDrawState);
begin
  if Table1Client.AsString = 'XXXX' then
  begin
    DBGrid.Canvas.Brush.Color := clRed;
    DBGrid.Canvas.Font.Color := clSilver;
    DBGrid.Canvas.FillRect(Rect); { Desenha o pano de fundo }
    DBGrid1.Canvas.TextOut(Rect.Left+2, Rect.Top+1, Field.AsString);
  end;
end;
```

6. Como chamar um outro programa (tipo o notepad do Windows[®]) a partir de um aplicativo (de três maneiras diferentes: normal, maximizado e minimizado):

```
WinExec('C:\windows\notepad.exe', SW_SHOWNORMAL);
WinExec('C:\windows\notepad.exe', SW_SHOWMAXIMIZED);
WinExec('C:\windows\notepad.exe', SW_SHOWMINIMIZED);
```

7. Como varrer uma tabela inteira:

```
Table1.First;
if not Table1.Eof then
  repeat
    { seus comandos para a tabela }
    Table1.Next
  until Table1.Eof;
```

8. Como interceptar as teclas de função:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F5 then
    ShowMessage('Você pressionou a F5');
end;
{ pode ser usado da VK_F1 a VK_F12 }
```

9. Como os valores de campos de uma tabela para outra:

```
{ Este exemplo copia apenas tabelas de mesma estrutura }
var
  Num: SmallInt;
begin
  for Num := 0 to TabelaOrigem.FieldCount - 1 do
    begin
      TabelaDestino.Insert;
      TabelaDestino.Fields[Num].Assign(TabelaOrigem.Fields[Num]);
      TabelaDestino.Post;
    end;
end;
```

10. Como verificar se um campo inteiro é par ou ímpar:

```
function TestaParaPar(TestaInteiro : Integer) : boolean;
begin
  if (TestaInteiro div 2) = (TestaInteiro/2) then
    result := True
  else
    result := False;
end;
```

11. Como verificar se uma string contém um inteiro:

```
function IsInteger(TestaString: String) : boolean;
begin
  try
    StrToInt(TestaString);
  except
    On EConvertError do result := False;
  else
    result := True;
  end;
end;
```

12. Como subtrair datas:

```
function SubData(DataEmprestimo: TDateTime) : Integer;
begin
  result := Date - DataEmprestimo;
end;
```

