

Índice

| | |
|--|----|
| INICIALIZANDO O DELPHI | 2 |
| CRIANDO UM PEQUENO PROJETO | 3 |
| OUTRAS FORMAS DE CRIAR UM PROJETO | 4 |
| LINGUAGEM OBJECT PASCAL - CONCEITOS BÁSICOS | 6 |
| PALAVRAS CHAVE..... | 6 |
| VARIÁVEIS..... | 7 |
| ARRAYS (VETORES)..... | 8 |
| RECORDS (REGISTROS)..... | 8 |
| CLASSES E OBJETOS..... | 9 |
| COMPONENTES, CONTROLES E PROPRIEDADES..... | 9 |
| FUNÇÕES | 9 |
| PROCEDIMENTOS..... | 10 |
| PASSAGEM DE PARÂMETROS | 10 |
| MÉTODOS E EVENTOS | 11 |
| ESTRUTURAS DE CONTROLE EM OBJECT PASCAL..... | 11 |
| EXEMPLO DO USO DE FUNÇÕES E PROCEDIMENTOS..... | 13 |
| ESCREVENDO UM EDITOR DE TEXTO SIMPLES | 16 |
| APERFEIÇOAMENTOS DO EDITOR..... | 17 |
| COMPONENTES DE DADOS..... | 19 |
| PROJETO DE SISTEMA I | 19 |
| CRIAÇÃO DO PROJETO..... | 21 |
| OPERAÇÕES COM REGISTROS E CAMPOS (Componentes TField)..... | 24 |
| PROCURANDO DADOS: | 24 |
| OUTRAS FORMAS DE ACESSAR CAMPOS..... | 25 |
| FILTRAGEM DE REGISTROS : | 25 |
| A PROPRIEDADE KEYEXCLUSIVE DE TTABLE | 25 |
| HABILITANDO E DESABILITANDO A VISUALIZAÇÃO DE DADOS..... | 26 |
| NAVEGANDO PELO DATASET :..... | 26 |
| PROPRIEDADES BOF e EOF..... | 26 |
| MODIFICANDO REGISTROS : | 26 |
| MARCANDO UM REGISTRO | 27 |
| DE VOLTA AO PROJETO PESSOAL..... | 27 |
| CÁLCULO DA FOLHA DE PAGAMENTO..... | 28 |
| PROJETO DE SISTEMA II..... | 29 |
| CRIAÇÃO DO BANCO DE DADOS..... | 29 |
| CRIAÇÃO DO APLICATIVO..... | 31 |
| CRIAÇÃO DO FORMULÁRIO CADASTRO DE CLIENTES | 34 |
| CRIAÇÃO DO FORMULÁRIO PARA QUERY DE CLIENTES..... | 36 |
| CRIAÇÃO DO FORMULÁRIO CADASTRO DE MERCADORIAS | 41 |
| CRIAÇÃO DO FORMULÁRIO MOVIMENTO DE VENDAS..... | 43 |
| USANDO O INTERBASE..... | 44 |
| TESTANDO O BD..... | 49 |

INICIALIZANDO O DELPHI

- grupo de programas do Delphi, no Win98 tem formato similar ao mostrado abaixo :



Basta selecionar o ícone do Delphi 4 no grupo de programas acima e teclar *Enter* para acessar o Delphi.

Nota : Um duplo-clique com o mouse sôbre o ícone indicado teria o mesmo efeito.

Uma vez inicializado o Delphi apresentará a sua IDE (Ambiente de Desenvolvimento Integrado) conforme mostrado na página seguinte.

Nesse ambiente tem-se acesso a um editor de código (Code Editor) para a criação, edição e visualização do código dos programas e a uma série de ferramentas para a construção do Aplicativo.

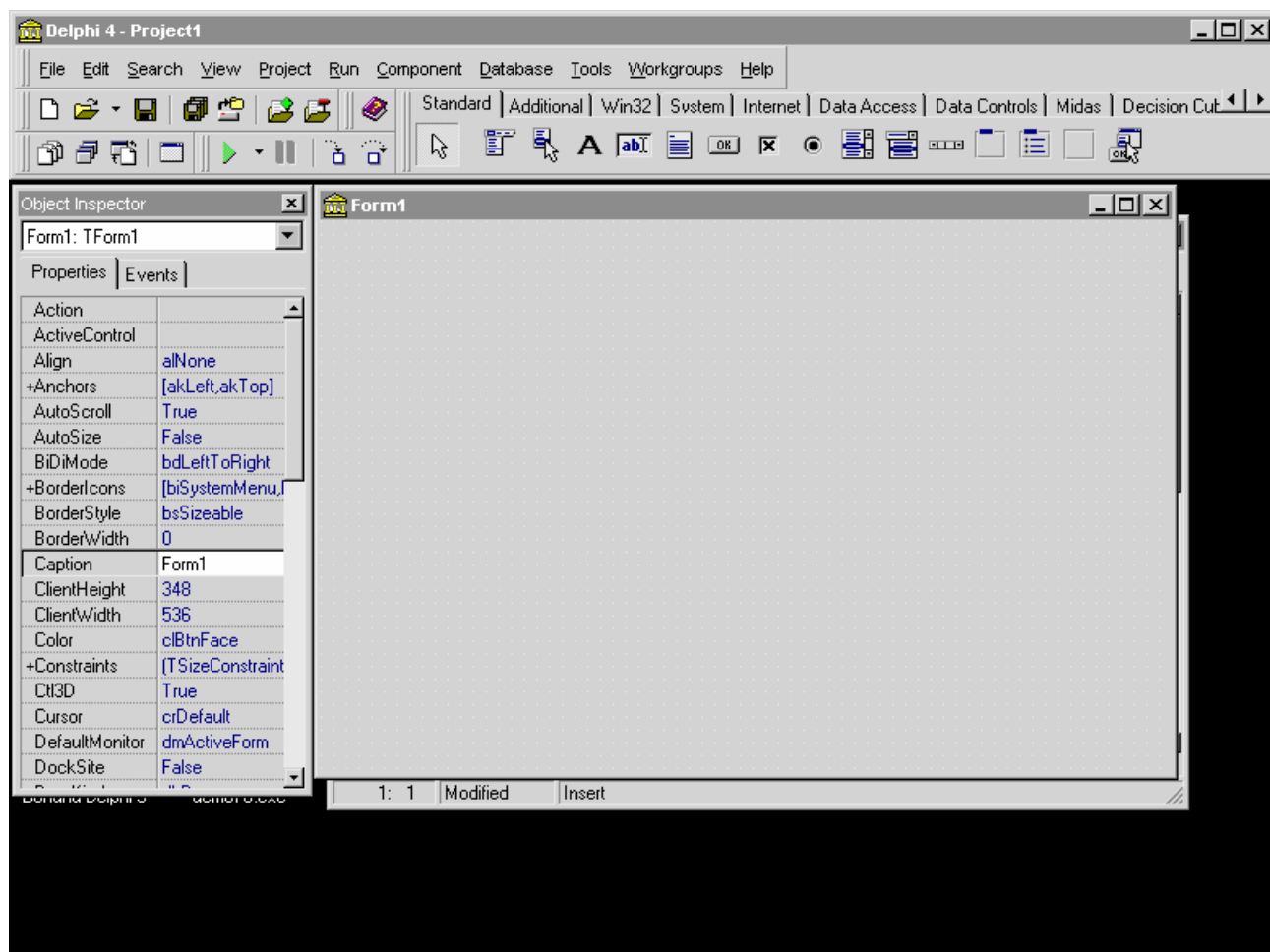
Observando-se a página seguinte, podemos identificar as seguintes áreas de trabalho :

- Uma barra de ferramentas contendo : a barra de títulos, a barra de menus, uma caixa de ferramentas e a paleta de componentes. Observe, na barra de título, que o Delphi atribui o nome *default* de *Project1* ao aplicativo.

- Uma janela com o nome de *Object Inspector* . Esta janela permite acesso direto as propriedades e eventos relacionados a um determinado componente.

- Uma janela na qual o Delphi coloca o nome *Form1*. Este é o formulário (ou forma, ou janela) criado automaticamente pelo Delphi. Por trás da janela está presente uma janela para a edição de código (Code Editor).

Iremos apresentando o funcionamento de cada um dos elementos da IDE do Delphi a medida que eles forem se tornando necessários.



CRIANDO UM PEQUENO PROJETO

Ao iniciar, o Delphi já criou e colocou a nossa disposição um projeto chamado *Project1*, com uma unidade de código *Unit1*. Vamos construir o nosso primeiro projeto em cima desta base inicial fornecida pelo Delphi. Poderíamos também escolher a opção *New Application* no menu *File* para criar o nosso primeiro projeto. Utilizaremos esta segunda modalidade em lições futuras.

Nosso projeto deverá ter o nome *primeiro.dpr* e seu único formulário deverá ter o nome *forma.pas*.

Ele deverá abrir uma janela com o título (*caption*) 'Primeiro Projeto' e, exibir um rótulo (*Label*) com a seguinte mensagem: 'Este é meu primeiro programa em Delphi'.

Para a confecção do projeto, siga os seguintes passos:

- crie, utilizando o Windows Explorer, um diretório(pasta) com o nome *CDelphi*. Este diretório deverá ser utilizado para todos os nossos trabalhos neste curso.

- crie, dentro de *CDelphi* um subdiretório com o nome *Ex01*.

- selecione (clicando com o mouse), na paleta *Standard* de componentes, o componente *Label* (o terceiro, a partir da esquerda). Clique sobre o formulário em qualquer posição, para inserir o componente selecionado. No *Object Inspector*, procure a propriedade *caption* e insira, na coluna a direita o texto: 'Este é o meu primeiro programa em Delphi'. O componente *Label* está selecionado e você poderá movimentá-lo

clicando e arrastando-o. Você pode também, redimensioná-lo utilizando as alças existentes com esta finalidade. Para centralizar o texto na janela, clique o botão direito do mouse e acesse a opção *Align* no menu que se apresenta. Selecione *center in window* na horizontal e na vertical e clique OK. *Selecionando o componente com um duplo clique também coloca-o no centro do formulário, mas a centralização pode ser alterada ao colocarmos o texto.*

- selecione o formulário (clicando em que qualquer ponto do mesmo (fora do rótulo criado anteriormente) e, no *Object Inspector* propriedade *caption* insira : 'Primeiro Projeto' e na propriedade *Name* insira *FormaPrincipal*.

Vamos escolher a opção *Save Project As...* no menu *File* . O Delphi vai nos solicitar primeiramente, um nome para o nosso módulo de código. Vamos salvá-lo (dentro do diretório c:\CDelphi\Ex01) com o nome *forma.pas*. Em seguida, o Delphi habilitará a digitação do nome do projeto. Vamos salvá-lo com o nome *primeiro* (que será também, o nome passado para o arquivo executável pelo Delphi).

Falta-nos apenas compilar o nosso projeto.

Temos diversas opções para a compilação do nosso projeto :

- no menu *Project* opção *Compile* (ou teclando Ctrl+F9) ; neste caso serão compilados apenas os módulos que sofreram alteração após o último processo de compilação.

- no menu *Project* opção *Build All* para obter a compilação de todos os módulos, sem exceção.

Obs : Neste menu temos ainda uma opção para *Check* de sintaxe.

- no menu *Run* opção *Run* (ou teclando F9 ou clicando no ícone correspondente na barra de ferramentas) ; neste caso serão compilados os módulos que sofreram alteração e o programa será executado.

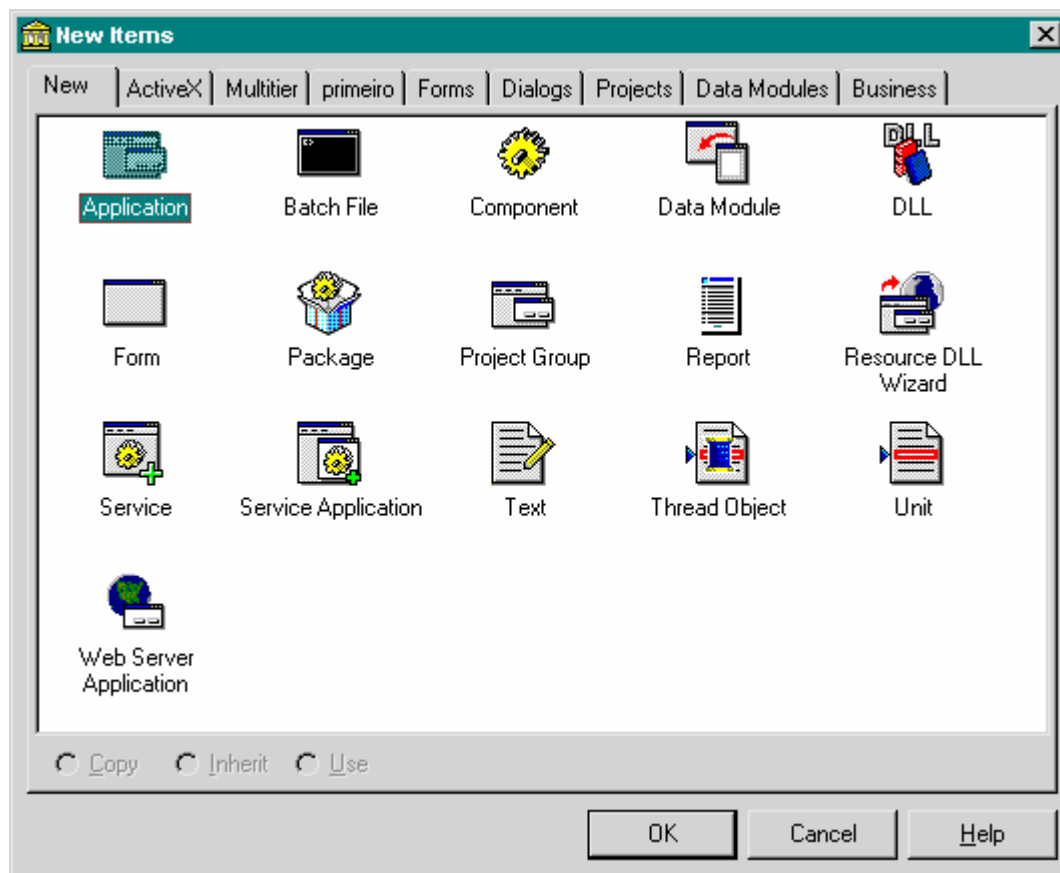
Normalmente, o uso da tecla F9 é o mais conveniente pois compilamos e comandamos a execução imediatamente...

Feita a compilação temos um programa completo e funcional. Ele não faz quase nada de produtivo, apenas abre uma janela do Windows e coloca uma mensagem no meio desta janela...

A título de exercício o aluno é convidado a verificar o funcionamento do programa : maximize e minimize o formulário; desloque-o e redimensione-o; feche o formulário utilizando o menu de sistema (e as teclas Alt+F4). Observe que a centralização do componente se perde ao redimensionarmos o formulário.

OUTRAS FORMAS DE CRIAR UM PROJETO

- Delphi oferece uma flexibilidade muito grande na criação de nossos projetos. Selecionando-se *New* no menu *File* será aberta a caixa *New Items* mostrada abaixo :



A caixa de diálogo New Items fornece uma vista interna do Repositório de Objetos do Delphi. Este repositório contém formulários, projetos e *wizards*. Você pode usar os objetos diretamente, copiá-los para dentro de seu projeto ou herdar itens de objetos existentes.

LINGUAGEM OBJECT PASCAL - CONCEITOS BÁSICOS

Vamos abrir o projeto primeiro.dpr criado anteriormente, utilizando o menu *File* opção *Open...* (ou *Open Project* ou *ReOpen*) e, em seguida vamos acessar o *Code Editor*. O *Code Editor* pode ser acessado de qualquer uma das seguintes maneiras :

- teclando Shift+F12 e selecionando o formulário desejado (quando houver mais de 1).
- teclando F12; esta tecla nos permite passar da visualização do formulário para a visualização do código e vice-versa e deve ser a preferida pelo aluno por ser a mais conveniente.
- clicando em um dos itens correspondentes na caixa de ferramentas.
- selecionando a opção *Project Manager* no menu *View*.

Observe que muitas linhas terminam com ponto e vírgula, esta é a forma que o Object Pascal usa para indicar o término de um comando. Você pode escrever mais de um comando em uma linha, esta não é, no entanto, uma boa prática pois o código fica confuso e difícil de depurar.

PALAVRAS CHAVE

Palavras-chave ou palavras reservadas são aquelas usadas pela sintaxe da linguagem com significado específico e, portanto, não podem ser usadas com outra finalidade. As palavras-chave aparecem em negrito no *Code Editor*.

Em nosso módulo de código *forma*, encontramos as seguintes palavras-chave, exibidas em negrito :

- **unit** : esta palavra-chave define o nome da unidade de código. Este nome será inserido na cláusula *uses* de outras unidades que precisem fazer referência a esta unidade.
 - **interface** : define o início de um trecho de código que termina antes da palavra-chave **implementation**. Neste trecho se insere o código que poderá ser acessado por outras unidades. É aqui que se declaram as constantes, as variáveis, os tipos de dados, funções e procedimentos a serem utilizados por outras unidades de código.
 - **uses** : esta palavra especifica as unidades de código que serão acessadas por esta unidade.
 - **type** : com esta palavra-chave o Delphi define o início do trecho de código em que são definidos os tipos de variáveis e de classes criados pelo programa. O aluno pode observar que o Delphi já utilizou esta parte para declarar a classe *TFormaPrincipal*.
 - **private** : define os elementos de uma classe que não poderão ser acessados de fora da classe;
 - **public** : define os elementos de uma classe que poderão ser acessados de fora da classe.
 - **var** : define o início do trecho de código em que são declaradas as variáveis e objetos.
 - **implementation** : define o início do trecho de código em que são implementadas as funções e procedimentos declaradas no trecho iniciado pela palavra chave **interface**.
 - **end** : palavra-chave usada para encerrar um bloco de código. São blocos de código :
 - um bloco de comandos iniciado pela palavra chave **begin**, caso em que **end** deve ser seguida de um ponto-e-vírgula (;).
 - a definição de uma **unit**; neste caso a palavra **end** deve ser seguida de um ponto (.) e este será o final do arquivo.
- Exceção : antes da cláusula **else** em instruções condicionais **if-then-else** compostas, não se insere ponto (.) ou ponto-e-vírgula (;), utilizando-se apenas a palavra **end**.

VARIÁVEIS

Nossos dados são armazenados na memória do computador. Para que nós não tenhamos que nos referir a estes dados de forma direta, através de um endereço numérico difícil de memorizar, o compilador nos permite utilizar variáveis com esta finalidade. Escolhendo nomes sugestivos (mnemônicos) para nossas variáveis (tais como *nome*, *funcao*, *idade*, *salario*) facilitamos bastante a compreensão de nosso código.

Para que o Delphi possa usar nossas variáveis, devemos primeiro declará-las, isto é, informar o nome e o tipo desejados. Por exemplo : o comando a seguir declara *idade* como sendo uma variável do tipo inteiro (*integer*) :

```
idade : integer;
```

As variáveis inteiras podem assumir valores entre -32768 e +32767. Elas ocupam 2 bytes na memória. Assim sendo, a declaração acima faz com que o Delphi reserve 2 bytes para a nossa variável *idade*. Note que a declaração do tipo de uma variável, em princípio não lhe atribui valores. Um erro comum em programação é tentarmos ler valores de variáveis não inicializadas, ou às quais ainda não se atribuiu valores...

Damos a seguir uma lista dos tipos de variáveis mais comuns do Object Pascal com suas faixas de valores e o espaço ocupado em memória:

- BOOLEAN : Tipo lógico que pode assumir somente os valores TRUE ou FALSE e ocupa 1 byte de memória.
- BYTE : Tipo numérico inteiro, pode assumir valores numa faixa de 0 a 255, ocupa 1 byte.
- CHAR : Tipo alfa-numérico, pode armazenar um caractere ASCII, ocupa 1 byte.
- COMP : Tipo numérico real, pode assumir valores na faixa de $-9.2 \cdot 10^{-18}$ a $9.2 \cdot 10^{+18}$, ocupa 8 bytes, pode ter entre 19 e 20 algarismos significativos.
- EXTENDED : Tipo numérico real, pode assumir valores na faixa de $-3,4 \cdot 10^{-4932}$ a $+1,1 \cdot 10^{+4932}$, ocupa 10 bytes de memória e tem entre 19 e 20 algarismos significativos.
- INTEGER : Tipo numérico inteiro, pode assumir valores numa faixa de -32768 a +32767, ocupa 2 byte de memória.
- LONGINT : Tipo numérico inteiro, pode assumir valores numa faixa de -2147483648 a +2147483647, ocupa 4 bytes de memória.
- REAL : Tipo numérico real, pode assumir valores na faixa de $-2,9 \cdot 10^{-39}$ a $+1,7 \cdot 10^{+38}$, ocupa 6 bytes de memória e tem entre 11 e 12 algarismos significativos.
- SHORTINT : Tipo numérico inteiro, pode assumir valores numa faixa de -128 a +127, ocupa 1byte de memória.
- SINGLE : Tipo numérico real, pode assumir valores numa faixa de $-1,5 \cdot 10^{-45}$ a $+3,4 \cdot 10^{+38}$, ocupa 4 bytes de memória, e tem de 7 a 8 algarismos significativos.
- WORD : Tipo numérico inteiro, pode assumir valores numa faixa de 0 a 65535, ocupa 2bytes de memória.
- STRING : Tipo alfanumérico, possuindo como conteúdo uma cadeia de caracteres. O número de bytes ocupados na memória varia de 2 a 256, dependendo da quantidade máxima de caracteres definidos para a string. O primeiro byte contém a quantidade rela de caracteres da cadeia.

Os nomes de variáveis devem começar com uma letra ou o caractere sublinhado (_) seguido por uma sequência de letras, dígitos ou caractere sublinhado (_) e não podem conter espaço em branco nem quaisquer tipos de acentos. Os nomes de variáveis podem ter qualquer tamanho mas somente os 63 primeiros caracteres serão considerados.

Exemplos : Para definir uma variável *Nome* do tipo string e uma variável *Salario* do tipo *double*, podemos inserir as seguintes linhas de código na cláusula **var** da unidade de código correspondente.

```
Nome : string;  
Salario : double;
```

Pode-se declarar mais de uma variável do mesmo tipo na mesma linha, separando-as por vírgula.
nome, funcao, endereco : string;

ARRAYS (VETORES)

Arrays são conjuntos de variáveis com o mesmo nome e diferenciadas entre si por um índice. Eles são úteis para manipularmos grandes quantidades de dados de um mesmo tipo pois evitam a declaração de diversas variáveis.

Considere o caso de um programa de Folha de Pagamento que precise armazenar os seguintes dados referentes a 100 funcionários : nome, funcao, salário, etc... Seríamos obrigados a declarar 100 variáveis nome, 100 variáveis funcao, etc... O array nos permite declarar uma única variável com um índice para apontar para as diferentes ocorrências.

Declara-se um array da seguinte forma :

```
nome_da_variável : array[i1..i2] of tipo_de_variável; onde i1 e i2 representam os valores mínimo e máximo, respectivamente, do índice.
```

O Object Pascal permite que i1 e i2 possuam qualquer valor desde que i1 seja menor ou igual a i2. Assim, poderíamos declarar um array de 100 variáveis inteira *idade* de várias formas diferentes :

```
idade : array [1..100] of integer; ou  
idade : array [-100..-1] of integer; ou  
idade : array [0..99] of integer, etc...
```

Pode-se definir arrays multidimensionais (com vários índices) como, por exemplo :

```
espaco3d:array[1..10,-5..20,0..30] of double; que pode armazenar 10x26x31=8060 variáveis do tipo double.
```

Um dos casos mais comuns é a matriz com m linhas e n colunas : matriz : array[1..m,1..n] of qquer_tipo.

Os elementos dos arrays podem ser quaisquer tipos de variáveis ou objetos.

RECORDS (REGISTROS)

O Object Pascal permite definir tipos compostos de variáveis denominados registros. Define-se da seguinte forma :

```
nome_do_tipo : Record  
    variavel1 : primeiro_tipo;  
    variavel2 : segundo_tipo;  
    .....  
    variaveln : n-ésimo-tipo;  
end;
```

variavel1,variavel2.. variaveln são chamadas de campos do registro.

Declaramos uma variável deste tipo da mesma forma que procedemos para declarar variáveis de qualquer tipo pré-definido

```
variavel : nome_do_tipo;
```

Usamos a notação de ponto para acessar um campo de uma variável composta :

```
nome_da_variável.nome_do_campo;
```

Exemplo :

```
funcionario = Record  
    nome : string;  
    funcao : string;  
    salario : double;  
end;
```


Assim, no exemplo citado anteriormente, ao invés de declararmos um array nome de 100 elementos, um array funcao de 100 elementos, um array salario de 100 elementos, podemos declarar uma única variável chamada empregado, por exemplo, como sendo um array de 100 elementos do tipo funcionário.

```
empregado : array[1..100] of funcionario;
```

Para obter os dados do décimo funcionário, basta fazer :

```
empregado[10].nome;  
empregado[10].funcao;  
empregado[10].salario;
```

CLASSES E OBJETOS

Definição de classe :

```
nome_da_classe_derivada = class(nome_da_classe_base)  
  private  
  {propriedades, campos e métodos privados}  
  public  
  {propriedades, campos e métodos públicos}  
end;
```

A definição de uma classe é bastante similar a de um registro. As classes, diferentemente dos registros podem conter funções ou procedimentos, chamados métodos, além de variáveis.

Quando uma classe é derivada de outra (chamada classe base, classe pai ou classe mãe), ela herda os campos, propriedades e métodos da classe base.

O Delphi possui uma classe chamada TObject da qual todas as demais classes se derivam, ou seja, todas as classes são derivadas de TObject ou de classes derivadas de TObject. Quando você deriva uma classe de TObject, não é preciso declarar explicitamente a classe base pois o Delphi assume TObject como default.

Um objeto é uma instância de uma classe, isto é, depois de definirmos uma classe podemos declarar e utilizar objetos desta classe, da mesma forma que podemos declarar e usar uma variável composta (um registro, por exemplo) depois que o tivermos definido. De forma análoga, também, nos referimos aos elementos de um objeto utilizando a notação de ponto : nome_do_objeto.nome_do_elemento;

COMPONENTES, CONTROLES E PROPRIEDADES

O Delphi já possui muitas classes predefinidas para utilização no ambiente Windows. Algumas das classes do Delphi, chamadas genericamente de *componentes* podem ser acessadas através da paleta de componentes conforme já mostramos anteriormente.

Outra classe bastante utilizada é a classe TForm que define um formulário, mas que não está disponível na paleta de componentes. Mas, se olharmos na listagem do exercício Ex01 verificamos que o Delphi define automaticamente uma classe derivada de TForm (TFormaPrincipal no nosso exemplo) e declara um objeto desta classe (FormaPrincipal no nosso exemplo).

As propriedades são um tipo especial de campo em um objeto, cujo valor pode, em alguns casos, ser alterado usando-se o Object Inspector, como fizemos, por exemplo, com a propriedade Caption de FormaPrincipal no exercício Ex01. Outras propriedades só podem ser alteradas por programa (run-time). O sistema de Help do Delphi fornece, para cada classe, uma listagem completa de suas propriedades e seus métodos.

FUNÇÕES

Uma função se define em Pascal de forma semelhante a definição matemática. Ela recebe valores como parâmetros e retorna um outro valor como resultado. A definição de função obedece a seguinte sintaxe :

```
function nome_da_função(parâmetro_1:tipo_1,,,,,parâmetro_n:tipo_n) : tipo de retorno  
var  
{declaração de variáveis locais á função}  
begin  
{corpo da função}  
end;
```

O trecho de código a seguir define a função PRODXY, que retorna como resultado o produto de dois números reais X e Y:

```
function PRODXY(X,Y:Real):Real
begin
PRODXY := X*Y;
end;
```

Observe que o sinal de atribuição em Object Pascal é := e, não o sinal de igual simplesmente.

Em Object Pascal podem-se usar os seguintes operadores aritméticos :

+ : Soma;
- : Subtração;
* : Multiplicação;
/ : Divisão;
div : Divisão inteira e
mod : Resto da divisão inteira.

O Object Pascal tem várias funções predefinidas, parte das quais listamos a seguir :

Abs(x) Retorna o valor absoluto de x.
ArcTan(x) Retorna o valor do arco tangente de x (em radianos).
Cos(x) Retorna o cosseno de x (x em radianos).
Dec(x) Decrementa (subtrai 1) uma variável inteira x.
Exp(x) Retorna o valor de e elevado a x , onde e é a base dos logaritmos neperianos.
Frac(x) Retorna a parte fracionária do real x.
Inc(x) Incrementa (soma 1) uma variavel inteira x.
Int(x) Retorna a parte inteira do real x.
Ln(x) Retorna o logaritmo neperiano de x.
ODD(x) Retorna True se x for impar.
Sqr(x) Retorna o quadrado de x.
Sqrt(x) Retorna a raiz quadrada de x.

PROCEDIMENTOS

Um procedimento é semelhante a uma função, mas não retorna um valor. A chamada a um procedimento não pode ser colocada do lado direito de um comando de atribuição.

A definição de um procedimento obedece a seguinte sintaxe :

```
procedure nome_do_procedimento(parâmetro_1:tipo_1,,,parâmetro_n : tipo_n)
var
{declaração de variáveis locais ao procedimento}
begin
{corpo do procedimento}
end;
```

PASSAGEM DE PARÂMETROS

Podemos passar parâmetros para uma função ou um procedimento *por valor* e *por referência*. Quando se passa um parâmetro *por valor* estamos realmente passando um valor ou a cópia de um valor armazenado numa variável. Quando a passagem se faz por referência, estamos passando o endereço da variável na memória e não o seu valor. Quando se altera, dentro da função, o valor de uma variável passada por *por referência* esta alteração surte efeito em todo o programa (fora da função).

Para passarmos um parâmetro *por referência* devemos precedê-lo da palavra reservada **var**.

MÉTODOS E EVENTOS

Um método é um tipo especial de procedimento ou função, definido dentro de uma classe. Por ser definido dentro da classe, ele pode acessar diretamente os campos dela sem a necessidade de passar estes campos como parâmetros. Para executar um método de uma determinada classe basta usar a mesma notação de ponto usada para acessar um campo.

Um evento é muito semelhante a um método pois ambos são definidos internamente a uma classe. A diferença é que os eventos são executados em resposta a alguma ação do usuário ou em resposta a uma mensagem enviada pelo Windows. Cite-se como exemplo o evento OnClick de um formulário, um procedimento que é executado toda vez que o usuário dá um clique com o mouse sobre o formulário. Isto permite que o nosso programa execute alguma ação quando o usuário clica com o mouse sobre o controle.

ESTRUTURAS DE CONTROLE EM OBJECT PASCAL

Nos limitaremos a apresentar uma breve descrição da sintaxe dessas estruturas; os próximos exemplos utilizam-nas e servirão de exemplos de aplicação.

ESTRUTURA CONDICIONAL **if-then-else**

Sintaxe :

```
if (condição)
then
    begin
        {Bloco de comandos executados se a função for verdadeira}
    end
else
    begin
        {Bloco de comandos executados se a função for falsa}
    end;
```

Caso você não queira executar qualquer comando se a condição for falsa, suprima toda a cláusula **else** :

```
if(condição)
then
    begin
        {Bloco de comandos executados se a função for verdadeira}
    end;
```

Note que o **end** que precede a cláusula **else** não é seguido de ;.

Note que as cláusulas **begin** e **end** só são necessárias para blocos com mais de uma linha.

ESTRUTURA CONDICIONAL **case-of**

Sintaxe :

```
case expressão of
    constante_1_1,...constante_1_n : bloco_de_comando_1;
    constante_2_1,...constante_2_n : bloco_de_comando_2;
    ...
    constante_n_1,...constante_n_n : bloco_de_comando_n;
else
bloco_de_comando;
end;
```

O comando **case** é um substituto mais elegante e mais legível para **if-then-else** múltiplos. A expressão (ou seletor) deverá ser de tipo com o tamanho máximo de 2 bytes (**Byte, Char,,Word ou Integer**) .

Segue-se um exemplo :

```
case Ch of
```

```
'A'..'Z', 'a'..'z' : WriteLn('Letra');  
'0'..'9' : WriteLn('Digito');  
'+', '-', '*', '/' : WriteLn('Operador');  
else  
  WriteLn(' Caracter Especial');  
end;
```

ESTRUTURA DE REPETIÇÃO (**for-to, for-downto**)

Sintaxe :

```
for contador:=valor_inicial to valor_final do  
  begin  
    {bloco_de_comandos}  
  end;
```

onde :

contador é uma variável inteira;

valor_inicial é o valor inicial do contador, que deve ser um número inteiro;

valor_final é o valor final (para o qual o laço se encerrará) a ser assumido pelo contador, deve ser inteiro;

Se se desejar que o contador assuma valores decrescentes deve-se usar a seguinte sintaxe :

```
for contador:=valor_inicial downto valor_final do  
  begin  
    {bloco_de_comandos}  
  end;
```

Note que as cláusulas **begin** e **end** só são necessárias para blocos com mais de uma linha.

ESTRUTURA DE REPETIÇÃO **while-do**

Sintaxe:

```
while expressão_booleana do  
  begin  
    bloco_de_comando;  
  end;
```

Note que o bloco_de_comando é executado enquanto a expressão_booleana for verdadeira.

Note que como a expressão_booleana é avaliada antes da execução do bloco_de_comando, se a expressão_booleana for falsa o bloco_de_comando não será executado nenhuma vez.

Note que as cláusulas **begin** e **end** só são necessárias para blocos com mais de uma linha.

ESTRUTURA **repeat-untill**

Sintaxe :

```
repeat bloco_de_comando untill expressão_booleana;
```

Note que o bloco de comando é executado em sequência, enquanto a expressão_booleanda for verdadeira.

Note que o bloco de comando é executado pelo menos uma vez, já que a expressão_booleana é avaliada depois dele.

Note que bloco_de_comando pode exigir as cláusula **begin** e **end** se contiver mais de uma linha.

ESTRUTURA with

Sintaxe

with identificador_de_registro(objeto) **do** bloco_de_comando

Note que **with** é uma forma abreviada de referenciar os campos de um registro ou os campos e métodos de um objeto. Dentro do bloco_de_comando (com **with**) uma variável ou método fica completamente identificada usando-se apenas seu identificador de campo.

ESTRUTURA try-except-end

Sintaxe :

try

sequência_de_comandos

except

bloco_a_ser_executado_no_caso_de_erro

end;

Esta estrutura nos permite tratar de forma conveniente os erros ocorridos no programa. Sem ela, ocorrendo um erro, o Delphi dispara o seu mecanismo de tratamento de exceções, normalmente emitindo uma mensagem de erro em inglês e não nos dando chance de recuperação. Com esta estrutura podemos detectar o erro, fornecer uma mensagem inteligível e permitir o retorno à rotina onde ocorreu o erro e permitir, por exemplo, a possibilidade de digitação.

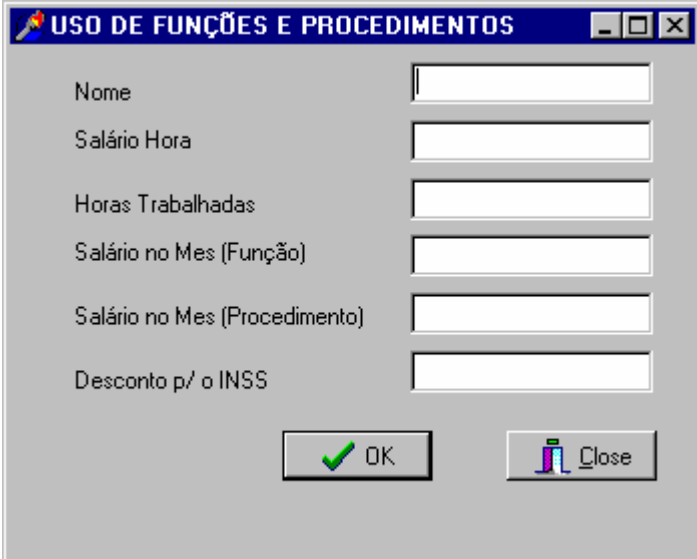
EXEMPLO DO USO DE FUNÇÕES E PROCEDIMENTOS

Vamos criar, no Delphi, um projeto com um formulário em branco; vamos dar ao projeto o nome de segundo.dpr e ao formulário (que é único), o nome forma.pas. Devemos criar um diretório C:\CDelphi\Ex02 e mandar salvar o programa neste diretório. Vamos dar o título (caption) de 'Uso de Funções e Procedimentos' ao nosso formulário.

Vamos baixar 5 componentes Tedit, 5 componentes TLabel e dois componentes BitBtn.

Façamos a propriedade Active Control da Form1 = a Edit1 e mudemos o caption de Form1 para : Funções e Procedimentos.

Vamos retirar os textos das caixas de edição e fazer os captions dos rótulos para que a nossa janela fique com o visual mostrado abaixo :



The image shows a screenshot of a Delphi form window titled "USO DE FUNÇÕES E PROCEDIMENTOS". The form has a light gray background and contains six text input fields arranged vertically. Each field is preceded by a label: "Nome", "Salário Hora", "Horas Trabalhadas", "Salário no Mes (Função)", "Salário no Mes (Procedimento)", and "Desconto p/ o INSS". At the bottom of the form, there are two buttons: "OK" with a green checkmark icon and "Close" with a red 'X' icon. The window title bar includes standard Windows controls (minimize, maximize, close).

Segue-se o código correspondente :

unit forma;

interface

uses

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

type

TFprincipal = class(TForm)
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
procedure BitBtn1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

Empregados = class

public

nome : string;
funcao : string;
salario : double;
salario_hora : double;
horas_trab : integer;
function calcula_salario : double;
{Declaramos uma função dentro da classe, isto é um método }
procedure desconto_inss ;
{Declaramos um procedimento dentro da classe, isto é um método }

end;

var

Fprincipal: TFprincipal;
Funcionario : Empregados;
taxa_inss : integer;

implementation*{\$R *.DFM}***procedure** calc_salario(var salario, salario_hora : double ; horas_trab : integer);**begin**

salario := salario_hora * horas_trab;

end;**function** Empregados.calcula_salario : double;**begin**

calcula_salario := Funcionario.salario_hora * Funcionario.horas_trab;

end;**procedure** Empregados.desconto_inss;**begin***{Isto é um método - não preciso da notação de ponto em salario. Também não preciso devolver valores, pois estou imprimindo o valor em Edit6 aqui dentro do procedimento. Edit6 não pertence a classe. Tenho que precede-la do nome do objeto que a possui (no caso FPrincipal) e um ponto}*

FPrincipal.Edit6.Text := FloatToStr(taxa_inss * salario/100);

end;**procedure** TFprincipal.BitBtn1Click(Sender: TObject);**begin**

Funcionario.nome := Edit1.Text;

Funcionario.salario_hora := StrToFloat(Edit2.Text);

Funcionario.horas_trab := StrToInt(Edit3.Text);

*{A função retorna um valor podemos usá-la a direita de um sinal de atribuição. Notar que não precisamos passar parâmetros para a função, pois a mesma pertence a classe (é um método da classe). Tivemos, sim, que usar a notação de ponto. }**{Teste1:}*

Funcionario.salario := Funcionario.calcula_salario;

Edit4.Text := FloatToStr(Funcionario.salario);

*{Poderíamos escrever também, de forma mais compacta :**Edit4.Text := FloatToStr(Funcionario.Calcula_salario);}**{A procedure não retorna valores, se ela precisar devolver um valor teremos que passar um parametro por referencia, no caso passamos Funcionario.salario. A procedure nada tem a ver com a classe, assim, temos que passar valores através de parâmetros. Também não usamos a notação de ponto no nome da procedure , pois ela é independente da classe.}*calc_salario(Funcionario.salario,Funcionario.salario_hora,
Funcionario.horas_trab);

Edit5.Text := FloatToStr(Funcionario.salario);

Funcionario.desconto_inss;

end;**procedure** TFprincipal.FormCreate(Sender: TObject);**begin**

Funcionario := Empregados.create;

taxa_inss := 10; *{Fizemos o desconto do inss = a 10% para todos}***end;****end.**

ESCREVENDO UM EDITOR DE TEXTO SIMPLES

Vamos criar, no Delphi, um projeto utilizando a template SDIApplication (File/New/Projects/SDIApplication). Vamos utilizar os nomes dados pelo Delphi. Devemos criar um subdiretório C:\CDelphi\Ex03 e mandar salvar o programa neste diretório.

Vamos baixar um componente TMemo ocupando toda a área do cliente. Para isso, dê um duplo clique no componente (na paleta de componentes); isto colocará o componente no meio da tela. Em seguida, no Object Inspector faça Align = alClient. Isto fará com que ele ocupe toda a área do cliente. Entre na propriedade Lines de Memo1 e apague a primeira linha colocada pelo Delphi. Observe a forma alternativa que usamos para baixar um componente. Isto pode ser usado com qualquer componente, que será sempre colocado no meio da tela. *Área do cliente* é a área útil da tela, isto é, descontadas a barra de título, a barra de menus, a barra de ferramentas e a barra de status.

Façamos a propriedade Active Control do formulário = Memo1 e mudemos o seu caption para : Editor de Texto. Vamos criar um procedimento para o evento OnShow de SDIAppForm. Para tal, no Object Inspector, em Eventos dê um duplo clique em OnShow. Insira, na procedure aberta pelo Delphi, o código :

```
Memo1.SetFocus;
```

Isto faz com que possamos digitar texto de imediato a partir do início de Memo1. Substitua trechos de código, como abaixo :

```
procedure TSDIAppForm.OpenItemClick(Sender: TObject);
begin
  if OpenFileDialog.Execute = True then
    Memo1.Lines.LoadFromFile(OpenDialog.FileName);
  Memo1.SetFocus;
end;
```

```
procedure TSDIAppForm.SaveltemClick(Sender: TObject);
begin
  if SaveDialog.Execute = True then
    Memo1.Lines.SaveToFile(SaveDialog.FileName);
end;
```

Vamos escolher um speedbutton (paleta adicional) para o menu Ajuda. Carregue C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Images\Buttons\Help.bmp na propriedade Glyph. Se tiver dificuldade em alinhar o botão na altura faça sua propriedade top igual a de um qualquer dos outros botões. Escolha para seu evento OnClick o mesmo do Item de menu About...

Coloque a propriedade FileName= *.txt. Default Extension = txt e Filter = Arquivo Texto (*.txt)|*.txt para os componentes OpenFileDialog e SaveDialog.

Vamos traduzir os menus e mudar rótulos do formulário About para que os formulários mostrem o visual abaixo :

NOTA : O icone foi escolhido com duplo clique no componente Image, selecionando Load e dando Ok na figura escolhida. Escolhemos C:\Arquivos de Programas\Borland\Delphi3\Images\icons\handshak.ico.

Altere os Hints da seguinte forma :

```
  Abrir|Abre arquivo existente
  Salvar|Salva o Arquivo
  Sair|Encerra o Programa
  Sobre|Autoria e Informações
```

O aluno deve observar :

```
  Uso dos hints nos botões da speedbar e nos itens de menu;
  Teclas sublinhadas no menu;
  Teclas de atalho;
  Uso de ... (no menu)
```

Nosso programa está pronto e pode ser compilado e testado pelos alunos...

APERFEIÇOAMENTOS DO EDITOR

Vamos inserir algumas opções para escolha de Fontes, Cor de Fundo e Seleções (de Comprimento de Linha e Atributos de Leitura/Gravação).

Para tal vamos baixar um componente FontDialog da paleta Dialogs; vamos inserir o item de menu Editar|Fontes (dando um duplo clique no componente Tmenu).

Dê um duplo clique no evento OnClick do item de menu Fontes1 e tecle na procedure aberta, o código abaixo :

```
if FontDialog1.Execute then
  Memo1.Font := FontDialog1.Font;
```

Vamos baixar um componente ColorDialog da paleta Dialogs e inserir o item de menu Editar|Cor de Fundo (dando um duplo clique no componente Tmenu).

Dê um duplo clique no evento OnClick do item de menu CordeFundo1 e tecle na procedure aberta, o código abaixo :

```
if ColorDialog1.Execute then
  Memo1.Color := ColorDialog1.Color;
  SDiAppForm.Color := Memo1.Color;
```

Note que fizemos a cor do formulário igual a cor do componente Memo1.

Vamos inserir novo formulário (selecionando New|Dialogs). Escolha o Standard Dialog com botoes na vertical. Salve este formulário com o nome de módulo Seleta.pas e nome de formulário FrmSel e coloque os componentes mostrados.

Observar que fizemos o primeiro grupo de Radio Buttons dentro de uma GroupBox e baixamos 3 RadioButtons. No segundo grupo preferimos baixar um RadioGroup e gerar os RadioButtos na propriedade Itens. No primeiro caso escolhemos o botão default com a propriedade Checked = True, no segundo caso fizemos ItemIndex = 0.O terceiro conjunto está dentro de um GroupBox. E a seleção também se faz pela propriedade Checked.

Vamos criar o item de menu Editar|Seleções, dar um clique duplo no evento OnClick e teclar o código que se segue, na procedure aberta pelo Delphi :

```
FrmSel.ShowModal;
if FrmSel.ModalResult=mrOK then
  begin
  if FrmSel.RadioButton1.Checked = True then
    Memo1.Width := SDiAppForm.Width;
  if FrmSel.RadioButton2.Checked = True then
    Memo1.Width := SDiAppForm.Width*3 div 4;
  if FrmSel.RadioButton3.Checked = True then
    Memo1.Width := SDiAppForm.Width div 2;
  if FrmSel.RadioGroup1.ItemIndex = 0 then
    Memo1.ReadOnly := True
  else
    Memo1.ReadOnly := False;
  if FrmSel.CheckBox1.Checked = True then
    Memo1.Alignment := taCenter
  else
    Memo1.Alignment := taLeftJustify;
  if FrmSel.CheckBox2.Checked = True then
    Memo1.WordWrap := True
  else
    Memo1.WordWrap := False;
```

```
end;
```

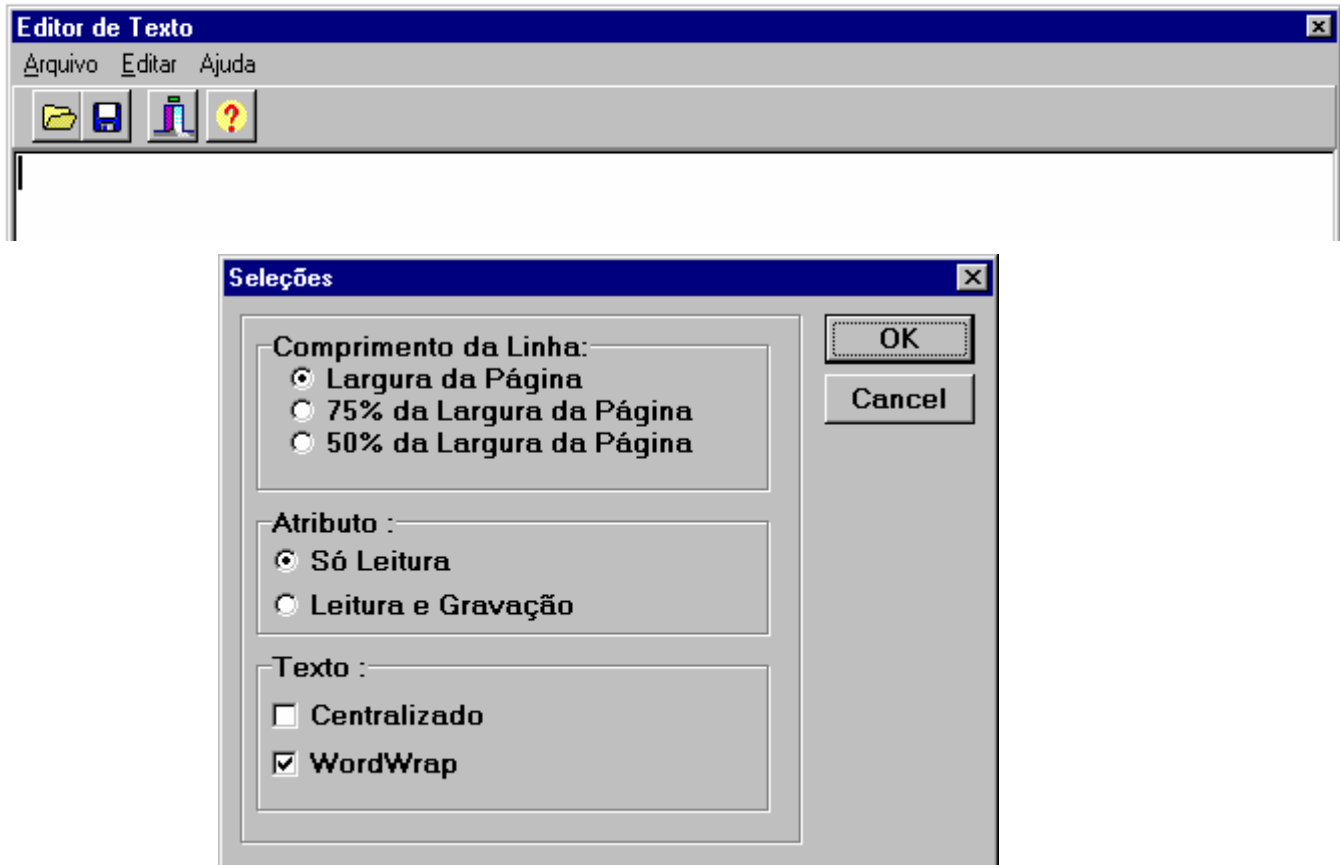
Insira Seleta na uses de SDIMain.

Refaça a propriedade Align de Memo1 para Align = alNone (para possibilitar a variação da largura).

Note poScreenCenter na propriedade Position do formulário. Verifique também seu Border Style (para não variar o tamanho da tela. Verifique FormCreate e FormShow p/ ajuste inicial de cor e uso de Hint. Notar 'bug' da cor e escondendo o bug de não variar o tamanho.

Note que ao mudarmos a fonte a mudança atinge tudo inclusive o texto já digitado. Esta é uma limitação do componente TMemo. O Delphi3 possui um component TRichEdit com propriedades bem superiores.

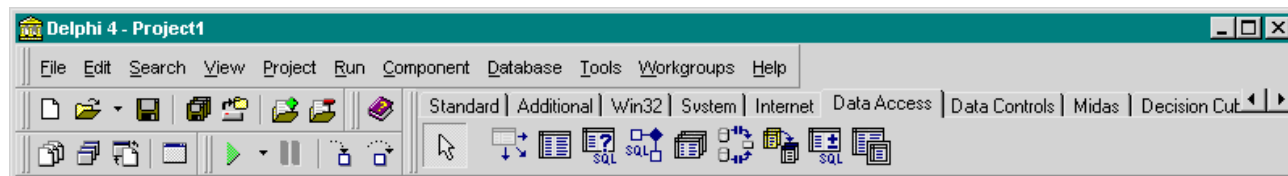
As figuras a seguir, mostram como ficou o visual do Editor e da tela de seleções.



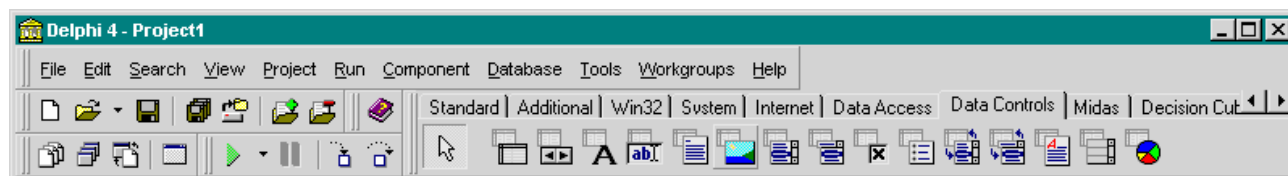
COMPONENTES DE DADOS

O Delphi oferece ao usuário (na sua Paleta de Componentes) duas páginas de componentes aplicáveis a bancos de dados :

- A página *Data Access* contém componentes que acessam os dados e permitem especificar o banco de dados, suas tabelas e registros. Clicando-se a aba correspondente a esta página, serão mostrados os componentes abaixo :
- Na ordem : DataSource, Table, Query, StoredProc, DataBase, Session, BatchMove, UpdateSQL, Provider, ClientDataSet, RemoteServer.



- A página *Data Controls* contém componentes de visualização e edição de dados. Eles nos permitem visualizar e entrar dados nas tabelas. Clicando-se a aba correspondente a esta página, serão mostrados os componentes abaixo :
- Na ordem : DBGrid, DBNavigator, DBText, DBEdit, DBMemo, DBImage, DBListBox, DBComboBox, DBCheckBox, DBRadioGroup, DBLookupListBox, DBLookupComboBox, DBRichEdit, DBCtrlGrid e DBChart.



Nota : O componente Data Source, apesar de incluído na página Data Access é, na realidade, um componente que faz a ligação entre os componentes de acesso e os componentes de visualização de dados.

PROJETO DE SISTEMA I

A melhor forma de se estudar o funcionamento dos vários componentes referentes a bancos de dados é utilizando-os em um projeto simples, como faremos a seguir :

Nosso projeto terá por finalidade processar a Folha de Pagamento de uma pequena empresa e terá duas tabelas, 'TabFun.db', contendo dados cadastrais dos funcionários da empresa e 'TabDados.db' contendo os 'dados variáveis' de cada funcionário para um mês. Esta última tabela contém um registro para cada semana trabalhada contendo o número de horas que o funcionário trabalhou na semana. Os funcionários são horistas, isto é, recebem por hora trabalhada. O pagamento é mensal e seu cálculo envolve totalizar as horas no mes (somando as horas na semana) e multiplicar este total pelo salário hora. Antes de iniciarmos o projeto vamos criar as tabelas do sistema. Devemos, também, preliminarmente, criar um subdiretório independente C:\CDelphi\Dados para conter as nossas tabelas.

Tabela 'TabFun.db' :

- Entre *Tools/Database Desktop* e escolha no menu *File*, a opção *Working Directory = C:\CDelphi\Dados* (isso facilitará o nosso trabalho posterior). Em seguida, ainda em *File* escolha *New*, sub-opção *Table*.
- No quadro *Table Type* escolha tabela do tipo *Paradox 7* e, no quadro *Create Table*, crie os seguintes campos :

Dicas de operação : Use a tecla ENTER para passar de uma coluna para outra; estando na coluna Type tecla 'barra de espaço' para baixar uma lista com os tipos disponíveis ; o " * " na coluna Key indica que queremos que seja um campo chave ; estando na coluna Key tecle qualquer caractere para setar ou resetar " * " .

'Required field' indica tratar-se de um campo de preenchimento obrigatório; Min e Max são os valores mínimo e máximo que o campo poderá assumir; estas informações permitem ao BD, independentemente do programa, verificar a validade dos dados fornecidos (validity checks).

| FieldName | Type | Size | Key | Required | Min | Max |
|--------------|------|------|-----|----------|-----|------|
| Matricula | A | 4 | * | Sim | 1 | 9999 |
| Nome | A | 40 | | Sim | | |
| Funcao | A | 20 | | | | |
| Sexo | A | 1 | | | | |
| Idade | S | | | | 16 | 70 |
| Salario_hora | \$ | | | | | |
| Horas_No_Mes | S | | | | 0 | 220 |

Salve a tabela com o nome *TabFun* clicando no botão *Save As*.

Note que apesar de escolhermos Paradox 7, o DBDesktop utilizou Paradox 4. Isto é automático e significa que não utilizamos tipos de campos das versões mais modernas.

A tabela *TabFun* foi criada nos passos anteriores, mas está vazia. Vamos colocar alguns dados nela :

- No menu *File*, opção *Open*, sub-opção *Table*, abra a tabela *TabFun*; em seguida, no menu *Table*, escolha a opção *Edit Data*, para trabalhar no modo de edição.
- Cadastre os seguintes funcionários :

| MATRICULA | NOME | FUNÇÃO | SEXO | IDADE | SALARIO_HORA |
|-----------|-----------------|--------------|------|-------|--------------|
| 1 | JOSÉ S. MELO | PROGRAMADOR | M | 22 | 20 |
| 2 | MAURÍCIO MAUROS | DIGITADOR | M | 21 | 15 |
| 3 | LUCIANA LUCIA | PROGRAMADORA | F | 25 | 22 |
| 4 | MARLUCIA LUCIA | ANALISTA | F | 30 | 32 |
| 5 | FERNANDO FARO | ANALISTA | M | 33 | 30 |

Obs: O campo HORAS_NO_MES será preenchido por programa, quando necessário.

Tabela '*TabDados.db*' :

Utilizando procedimento similar ao anterior, gere esta tabela com os seguintes campos :

| FieldName | Type | Size | Key | Required | Min | Max |
|-----------------|------|------|-----|----------|-----|------|
| Matricula | A | 4 | * | Sim | 1 | 9999 |
| Semana | S | | * | Sim | 1 | 5 |
| Horas_na_Semana | S | | | | 0 | 48 |

Preencha alguns dados como abaixo :

| MATRICULA | SEMANA | HORAS_NA_SEMANA |
|-----------|--------|-----------------|
| 1 | 1 | 30 |
| 1 | 2 | 35 |
| 1 | 3 | 44 |
| 1 | 4 | 38 |
| 2 | 2 | 20 |
| 2 | 4 | 33 |
| 3 | 1 | 40 |
| 3 | 3 | 44 |
| 3 | 4 | 45 |

CRIAÇÃO DO PROJETO

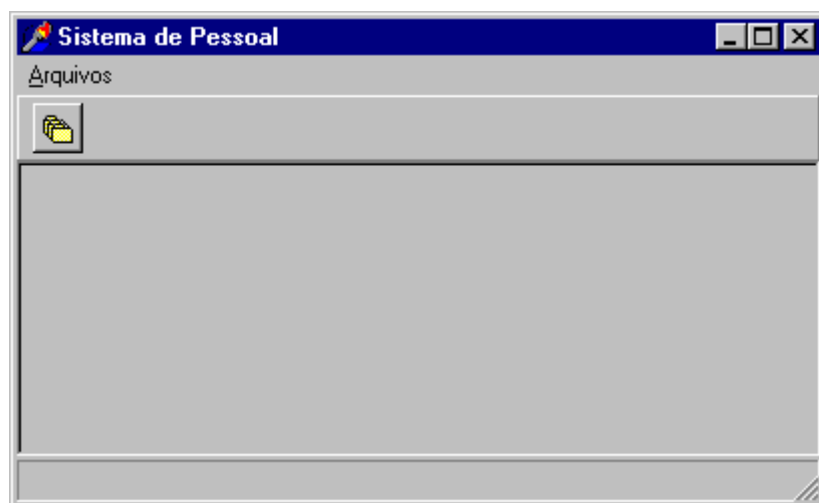
Selecione menu *File|New*; selecione a página *Projects* e, nesta :

- selecione o *Application Wizard* (clicando *OK*).
- selecione as opções de menu *File menu* e *Help menu*.
- Clique *Next* para prosseguirmos com o projeto;
- Clique *Next* pois não usaremos a próxima opção...
- Em seguida faça as seguintes seleções de speed bars :
 - selecione *File | Open* e clique *Insert*.
 - selecione *Help | About* clique *Insert* ; clique *Next*;

Faça o nome do projeto = *Pessoal* e o path = *C:\CDelphi\Ex04*.

Selecione as opções : *Create Status Line* e *Enable Hints*.

Após clicar em *Finish* será criado o aplicativo, conforme abaixo :



- Nota : mudamos o caption do formulário para 'SISTEMA DE PESSOAL'.

O aluno pode observar que o Wizard criou um menu *File* e um menu *Help* com vários sub-itens de menu. Nós queremos apenas um ou dois sub-itens em cada menu. O trabalho exigido para modificar os itens de menu inseridos pelo Wizard as vezes, é maior do que simplesmente deletar todo o menu e criar um novo. Vamos, por isso, deletar o do Delphi e criar um novo menu. Nosso menu deverá conter o item *&Arquivo* e os seguintes sub_itens , *&Cadastramento*, *&Funcionários* , *&Dados Variáveis*, 1 espaçador e *&Sair*, o restante deverá ser deletado. Deletar também os componentes que deixaram de ser usados : *TPrintDialog*, *TOpenDialog*, *TsaveDialog*, *TPrinterSetupDialog* e o 2º *SpeedButton*. O primeiro *SpeedButton* deve ser conservado, será usado na opção de menu *Cadastramento*.

Não esqueça de colocar os Hints nos itens de menu e no *SpeedButton* (mudamos o gliph do speedbutton para *C:\.Arquivos de Programas\Arquivos Comuns\Borland Shared\Images\Buttons\FLDRMANY.BMT*).

Apague o código de todas as procedures, menos as seguintes :

```
procedure FormCreate(Sender: TObject);  
procedure ShowHint(Sender: TObject);
```

Nota : basta apagar o código entre os begins e ends de cada uma e mandar salvar (Ctrl+S) que o Delphi se encarrega de deletar tanto na Implementation quanto na Interface. Conforme comentado anteriormente, teria sido mais rápido termos começado o projeto sem nenhum menu...

Antes de prosseguirmos, vamos criar um Alias para nosso Banco de Dados. Um Alias é um nome fictício que serve para designar um caminho de diretório onde colocamos nosso BD. O usuário, futuramente, pode colocar o BD onde melhor lhe aprouver, sem necessidade de se recompilar o programa.

Vamos usar o *DataBaseDesktop//Tools//Alias Manager//New*.

Façamos *Alias = Pessoal//Drive Type = Standard // Path = C:\CDelphi\Dados*

De Ok para encerrar e Ok outra vez para salvar as configurações novas no arquivo .CFG.

Vamos criar um formulário para cadastramento de funcionários e dados variáveis. Vamos utilizar para isso o Form Wizard (Database/FormWizard), com as seguintes opções :

- Master/Detail usando TTable
- Alias = Pessoal //Master Table = TabFun
- Escolha dos Campos (todos) >>
- Arranjo em grade
- Alias = Pessoal //Detail Table = TabDados
- Escolha dos Campos(todos) >>
- Arranjo em Grade
- Escolha Matricula nos dois lados e clique Add (campos relacionados).
- Gerar Form e DataModule (Não e' MainForm).
- Clicar Finish.

Mudar os nomes do DataModule para Dados e UnDados.pas.

.Poderíamos colocar nosso banco de dados no próprio formulário que o usa mas é boa prática termos os componentes de acesso a dados em um módulo de dados próprio e o Form Wizard faz isso para nós.

Observe como ficou o Form2 e vamos fazer o seu caption = 'Cadastramento'.

Vamos, em Dados, fazer as propriedades Active de ambas as tabelas = True. Voltemos ao formulário Form2 e verificamos que os dados estão visíveis (já que abrimos as tables no projeto). Podemos agora dimensionar o formulário para que todos os campos fiquem visíveis. Vamos inserir neste formulário um BitBtn tipo Close.

Vamos colocar hints nos botões do Navigator clicando na propriedade Hints, abrindo-se o Editor de String List e digitando : Primeiro Registro ; Registro Anterior ; Registro Seguinte ; Ultimo Registro ; Incluir Registro ; Excluir Registro ; Editar Registro ; Gravar Registro ; Cancelar Operação ; Atualizar. Temos também, que fazer a propriedade ShowHints = True.

Note que o último botão do Navigator poderia ter sido ocultado pois em aplicações desktop ele é desnecessário. Para tal agiríamos na propriedade +VisibleButtons e faríamos nbRefresh = False.

Vamo digitar 'Form2.ShowModal' ;no evento OnClick do item de Menu Cadastramento, vamos digitar 'Close;' no evento On Click do item de menu Sair, vamos colocar a mensagem 'Função Não Implementada' nos outros itens do menu e vamos compilar e executar.

Observações :

- Utilizamos duas tabelas relacionadas pelo campo Matricula (índice nas duas tabelas). Cada tabela exhibe os dados em uma grade própria. Os dados exibidos na grade de Dados Variáveis (tabela Detalhe) restringem-se aqueles correspondentes ao funcionário selecionado na grade de Funcionários (tabela Master).
- Observe em UnDados, Table1 e Table2, as propriedades preenchidas pelo Form Wizard : DataBaseName, IndexFieldName, MasterFields, MasterSource e TableName. Observe também as propriedades DataSet dos dois DataSource. Observe o TNavigator e sua amarração com Table1 através da propriedade DataSource.
- A amarração entre as grades (componentes TDBGrid) e as tabelas (componentes TTable) é feita pelo componente TDataSource. Observe (no Object Inspector) que a propriedade DataSet de DataSource1 aponta para a tabela Table1 e que a de DataSource2 aponta para Table2. Observe também que a propriedade DataSource de DBGrid1 aponta para Dados.DataSource1 e que a de DBGrid2 aponta para Dados.DataSource2. O componente TDataSource serve, portanto, como um componente de ligação entre um componente de acesso a dados (TTable) e um componente de exibição de dados (TDBGrid). A notação de ponto Dados.DataSource.. é necessária porque os componentes TDataSource estão em um módulo diferente daquele que contém as grades.
- Ao fazer a propriedade Active = True nas tabelas, nós realmente abrimos as tabelas em tempo de projeto. Isto nos permite ver (durante o projeto) o primeiro registro do conjunto de dados. Isto é bastante útil pois nos permite checar se não há erros.
- A propriedade DataBaseName das tabelas pode conter : o caminho de diretório + o nome do Banco de Dados, se houver (é o caso dos BDs relacionais e C/S) ; o caminho de diretório que contém a tabela ; um alias representando um dos casos acima. A importância do alias é que o BD poderá ser colocado em qualquer diretório conveniente sem precisarmos mexer no código do programa e recompilá-lo. Bastará

- neste caso, reconfigurar o alias utilizando uma das ferramentas oferecidas pelo Delphi (BDE Administrator ou DataBaseDesktop).
- O relacionamento Mestre/Detalhe das duas tabelas é feito da seguinte forma : a propriedade MasterSource de Table2 aponta para DataSource1 (o que a relaciona como detalhe com Table1) e a propriedade MasterFields aponta para o campo Matricula que é o campo comum (de relacionamento). Note que não há obrigatoriedade de os campos terem o mesmo nome nas duas tabelas...
 - A coluna Matricula na tabela detalhe é supérflua já que será sempre igual a Matricula selecionada na tabela mestre. Poderíamos eliminá-la : entre no Editor de Colunas clicando com o botão direito do mouse na grade, selecione AddAllFields, depois selecione Matricula e Delete. Devemos fazer isto somente depois que tivermos feito os testes necessários e, mais importante, se formos utilizar a grade para entrada de dados deveremos, por programa, inserir a Matricula no campo respectivo. Por ora deixemos o nosso campo no lugar. Basta fechar o módulo (sem salvar) e reabri-lo em seguida.
 - Componente TDataModule (UnDados.pas) : em nosso exemplo simples e numa configuração desktop poderíamos ter colocado nossos componentes de dados no próprio formulário de cadastramento. O Delphi disponibiliza, no entanto, um componente próprio para Dados (TDataModule) e devemos usá-lo. Quando se pensa em processamento distribuído (em redes locais, por exemplo) o tráfego na rede (entre o servidor e os clientes) é uma consideração importante. Nos grande BDs relacionais ou C/S procura-se executar no próprio servidor o máximo de tarefas possível afim de diminuir este tráfego. A organização dos programas em módulos adequados é, sob este aspecto, muito importante.
 - A DBGrid pode ser configurada através do Editor de Colunas. Clique com o botão direito do mouse na DBGrid, escolha Columns Editor/Add All Fields. Podemos mudar para cada coluna : Título, Cor de fundo, tipo e cor de fontes, etc...Vamos por exemplo, fazer o fonte de Matricula para Negrito tam 12, mudar a cor de fundo do Nome para clAqua e fonte tam 12/Negrito, colocar a função em Itálico, mudar o título do Nome para Nome do Funcionário.

Prosseguindo, vamos criar dois formulários um para cadastramento de funcionários e outro para dados variáveis. Para o primeiro vamos utilizar o Form Wizard (Database/FormWizard), com as seguintes opções :

- SimpleForm usando TTable
- Alias = Pessoal //Table = TabFun
- Escolha dos Campos (todos) >>
- Arranjo Vertically
- Labels in the left.
- Gerar Form Only (Não e' MainForm).
- Clicar Finish.

Selecione todos os componentes que usam DataSource (DBEdits e Navigator) e digite (uma vez só) Dados.DataSource1 na propriedade DataSource. Delete a Table1 e o DataSource1 criados pelo Wizard. Copie os hints do Navigator do Form2 e cole no deste formulário; faça ShowHints = True. No evento FuncionriosClick (MainForm) substitua o comando de mensagem pelo seguinte :
Form1.ShowModal.

Para o segundo formulário o procedimento é inteiramente análogo. A tabela é TabDados. Teclle F9, observe e corrija os erros que serão apresentados :

- referências de Unit1 e Unit3.
 - Table1 inexistente.
- O erro de referências é corrigido automaticamente pelo Delphi. Para corrigir o outro delete o evento FormCreate (de Unit1 e Unit3), já que as tabelas estão com Active = True (em UnDados). Ao teclar F9 ocorrerá outro erro de referência que mandaremos o Delphi corrigir. Podemos recompilar e testar.

Verifique o bug com a tabela TabDados. A grade só mostra valores para o primeiro funcionário. Isto ocorre porque esta tabela é 'escrava' de Tabfun, ou seja, ela só mostra os registros equivalentes aos de Tabfun que não pode ser navegada. Há duas maneiras para resolver este problema. A primeira consiste em retirarmos, por código o relacionamento das duas tabelas (o que nos obrigaria a restabelê-lo quando dele precisássemos). A segunda,

que usaremos aqui , é criar um terceiro conjunto Dados.Table3/Dados.DataSource3 (sem relacionamentos) e utilizarmos este último conjunto para o nosso objetivo.

OPERAÇÕES COM REGISTROS E CAMPOS (Componentes TField)

O Delphi encapsula cada campo de uma tabela com um componente TField próprio (do tipo desejado), assim, no nosso projeto, foram usados TStringFields, TSmallIntField, TCurrencyField. Devemos usar o Fields Editor (Editor de Campos) para manipular os componentes Tfields, durante o projeto. Este editor nos permite, entre outras coisas, determinar quais campos queremos mostrar no formulário.

Para abriremos o Editor de Campos, basta dar um duplo clique sobre o componente TTable. Aparece o Fields Editor. Clique no botão direito do mouse e selecione Add Fields... para aparecerem os nomes dos campos disponíveis; Selecione o campo que você quer que apareça no formulário, dando um clique sobre ele e clique OK; proceda da mesma forma para os outros campos desejados (Add, seleção, OK...); caso desejado mude a ordem de disposição dos campos bastando, para isso clicar e arrastar o campo para a posição desejada. (isto depois que todos os campos desejados estiverem selecionados). No nosso caso selecione todos os campos. Observando o Object Inspector você notará que o Delphi incluiu varios objetos da classe Tfield. Feche o Fields Editor pelo menu de sistema.

Como se faz com qualquer outro componente você pode mudar propriedades de campos Tfield usando o Object Inspector. Observe, por exemplo, a propriedade Currency (dinheiro) = True no componente Table1Salario_Hora e verifique que o Delphi formata o campo para reais. Nós estabelecemos isto quando, na montagem da tabela fizemos tipo do campo = \$. Vamos, a título de exemplo, fazer neste campo a propriedade DisplayFormat = #,##0.00. Ao recompilarmos verificamos que a máscara do salário foi modificada.

O aluno deve fazer uma análise de varias propriedades em especial as possibilidades de formatação. Estando sobre uma determinada propriedade, deve-se teclar F1 para obter o Help correspondente.

As seguintes propriedades se destinam a formatação de campos :

- DisplayFormat : para especificar o formato de visualização do dado;
- EditMask : fornece uma máscara de edição.
- Display width e Alignment especificam numero de caracteres e alinhamento.

Pode-se usar o evento OnGetText para formatar valores de campo durante a programação. Esse evento ocorre sempre que o Delphi vai mostrar o conteudo do campo.

Para validar (por programa) a entrada de dados, utilize o evento OnValidate que é acionado sempre que o valor de um campo é modificado.

PROCURANDO DADOS:

Devemos usar o método *FindKey* para procurar por um determinado valor de um campo numa tabela. O campo precisa estar indexado para usarmos este método. Por exemplo :

```
Table1.FindKey(['Maria']) para um campo Nome que indexe a tabela Table1.
```

Para indices compostos de mais de um campo use :

```
Table1.FindKey(['valor1', 'valor2', 'valor3'])
```

FindKey é uma função e devolve um resultado booleano que será verdadeiro se o registro foi encontrado.

Uma alternativa ao FindKey é o método FindNearest que posiciona o cursor na chave especificada ou naquela imediatamente superior. É o método utilizado em buscas sequenciais.

Além de FindKey/FindNearest, as versões posteriores ao Delphi2 possibilitam o uso dos métodos Locate e Lookup. Locate posiciona o cursor na chave desejada; Lookup recupera valores de campos do registro pesquisado. Estes métodos podem ser usados com campos indexados ou não.

OUTRAS FORMAS DE ACESSAR CAMPOS

Você pode usar o método `FieldByName` para acessar o valor de um campo. Você passa o nome do campo como parâmetro.

Por exemplo :

```
Editn.Text := Tablen.FieldByName('Nome') atribui a caixa de edição Editn o valor do campo 'Nome' da tabela Tablen. (Nota : Nome não precisa ser campo indice).
```

Se o campo não for do tipo string, utilize a função de conversão `AsString`:

```
Editn.Text := Tablen.FieldByName('codigo').AsString;
```

Você também pode usar a propriedade `Fields` que utiliza como parametro o número de ordem do campo na tabela.

Por exemplo , para atribuir a caixa de edição `Editn` o valor do campo `Nome`, que é o terceiro campo na tabela `Tablen` você faz

```
Editn.Text := Tablen.Fields[2]; { O primeiro campo tem o índice 0}  
ou Editn.Text := Tablen.Fields[n].AsString; para um campo que não seja do tipo string.
```

Nota : Neste método corre-se o risco de bugs se a estrutura da tabela for modificada já que neste caso a ordem e a posição dos campos pode mudar.

FILTRAGEM DE REGISTROS :

Os métodos abaixo (aplicáveis apenas a campos indexados para tabelas do tipo `Dbase` e `Paradox`) permitem filtrar registros de uma tabela :

```
SetRangeStart ; estabelece o limite inicial do filtro;  
SetRangeEnd : estabelece o limite final do filtro;  
ApplyRange : aplica o filtro a tabela;  
CancelRange : cancela o filtro aplicado a tabela.
```

Como alternativa pode-se usar :

```
SetRange([ValorInicial],[ValorFinal]) : estabelece os valores inicial e final do filtro e aplica a tabela;
```

Exemplo :

```
Tablen.SetRangeStart;  
Tablen.FieldByName('Codigo') := 20;  
Tablen.SetRangeEnd;  
Tablen.FieldByName('Codigo') := 60;  
Tablen.ApplyRange;
```

O exemplo permite filtrar todos os registros com código > 20 e menor que 60;

Você pode usar os métodos `EditRangeStart` e `EditRangeEnd` para mudar valores atribuídos anteriormente.

Você pode usar índices múltiplos para filtrar seus registros,. O exemplo a seguir vai filtrar todos os registros com código > 20 e nome igual a Maria.

```
Tablen.SetRangeStart;  
Tablen.FieldByName('Codigo') := 20;  
Tablen.FieldByName('Nome') := 'Maria';  
Tablen.SetRangeEnd;  
Tablen.ApplyRange;
```

As versões mais recentes do Delphi permitem algumas alternativas adicionais de filtragem, utilizando as propriedades `Filter` e `Filtered` e o evento `OnFilterRecord`.

A PROPRIEDADE KEYEXCLUSIVE DE TTABLE

A propriedade `KeyExclusive` é falsa por default, isto é, o filtro irá procurar valores maiores ou iguais ao do limite inicial e menores ou iguais ao do limite final.

Se você quiser (no exemplo com os limites 20 e 60 para o código) que o valor 20 não faça parte do filtro, (ou seja `Codigo > 20`), faça , por exemplo :

```
Tablen.SetRangeStart;  
Tablen.FieldName('Codigo') := 20;  
Tablen.KeyExclusive := True;  
Tablen.SetRangeEnd;  
Tablen.FieldName('Codigo') := 60;  
Tablen.ApplyRange;
```

HABILITANDO E DESABILITANDO A VISUALIZAÇÃO DE DADOS

Antes de iniciar um filtro pode ser importante você desabilitar os componentes de visualização inseridos no formulário, já que o filtro afeta todo o conjunto de dados (dataset). Esta providência é importante para se ganhar tempo, evitando que os componentes de Visualização fique mostrando dados provisórios e fique com flickering. Use estes controles com cuidado pois o usuário espera que os controles de visualização mostrem a realidade continuamente, assim, tendo desabilitado os controles, trate de habilitá-los logo que possível.

Faça `Tablen.DisableControls` antes de aplicar o filtro e

`Tablen.EnableControls` após a aplicação do filtro.

Deve-se usar um bloco `try ... finally` e comandar `Enable` dentro da clausula `finally`, senão, se ocorrer uma exceção os componentes de visualização não serão retomados.

NAVEGANDO PELO DATASET :

`First` : move o ponteiro para o primeiro registro

`Last` : move o ponteiro para o ultimo registro

`Next` : move o ponteiro para o próximo registro

`Prior` : move o ponteiro para o registro anterior

`MoveBy` ; move para a frente (se parametro positivo) ou para trás (se parâmetro negativo) um número especificado de registros , por exemplo : `MoveBy(2)`, avança dois registros.

`Next` equivale a `MoveBy(1)` e `Prior` equivale a `MoveBy(-1)`

PROPRIEDADES BOF e EOF

`BOF` = `True` indica que o ponteiro está no primeiro registro da tabela e ocorre nos seguintes casos :

- quando você abre a tabela;
- quando você executa o método `First`;
- quando você executa o método `Prior` e ocorre uma falha....

`EOF` = `True` indica que o ponteiro está no último registro da tabela e ocorre nos seguintes casos :

- Quando você abre um dataset vazio.
- Quando você chama o método `Last`;
- Quando você chama o método `Last` e ocorre falha no índice.

Exemplo :

```
while not Tablen.EOF do  
begin  
{ Faça algum processamento no registro}  
Tablen.Next;  
end;
```

MODIFICANDO REGISTROS :

Utilize `Edit` para possibilitar a modificação de dados no registro, em seguida utilize `Post` para gravar os novos dados no registro :

```
Tablen.Edit;  
{Comando de edição}  
Tablen.Post;
```

Utilize Insert para inserir um registro após o registro corrente;

Utilize Append para inserir um registro no final do arquivo;

Obs: Ao comandar Post, se houver um índice, o registro será inserido na posição dada pelo índice.

Utilize Delete para deletar um registro;

Utilize Cancel para cancelar alterações feitas (antes de comandar Post);

Utilize Refresh para apagar os buffers locais e buscar os dados originais na tabela (arquiv). Este comando pode aparentar resultados inesperados por exemplo : se um usuário está vendo um registro e um segundo usuário (em compartilhamento) deleta o registro, parecerá que o registro desapareceu quando o aplicativo usar Refresh.

Utilize a propriedade CanModify (de Ttable) para permitir ou proibir modificações no dataset . Se = True os dados podem ser alterados.

MARCANDO UM REGISTRO

Para marcar um registro e permitir que, após várias operações se volte a ele, utilize :

GetBookMark : marca o registro associando a ele uma propriedade do tipo Tbookmark;

GoToBookMark : leva o ponteiro até o registro marcado;

FreeBookMark : desmarca o registro.

DE VOLTA AO PROJETO PESSOAL

Após a digressão que fizemos, podemos voltar ao nosso projeto :

O editor de colunas permite várias outras operações interessantes. Voltemos ao Form2.

Vamos deletar a coluna Matricula na Grid2 pois ela é redundante (será sempre a mesma selecionada na Grid1). Selecione este campo no editor de colunas e use a opção Delete.

O campo Horas_no_mes da Grid1 é para uso do programa e não pode receber dados digitados pelo usuário. Vamos fazer no ObjectInspector : ReadOnly = True e ButtonStyle = cbsEllipsis (com Horas_no_mes selecionado no ColumnsEditor) Vamos abrir o evento OnEditButtonClick da Grid1 e inserir a seguinte mensagem de advertência para o caso do usuário tentar digitar neste campo.

```
MessageDlg('Este campo será preenchido'+#13+  
'pelo programa',mtInformation,[mbOK],0);
```

Para o campo Função, vamos apresentar uma lista na qual o usuário possa selecionar uma das possíveis funções existentes na empresa. Para tal criamos a tabela TabFunc.DB com os campos Codigo A//4 e Descricao A//20 e preenchemos com as seguintes funções : 1 – GERENTE; 2 – SUB-GERENTE; 3 – ANALISTA; 4 – PROGRAMADOR/A; 5 – DIGITADOR/A ; 6 – OPERADOR/A. No evento OnShow do Formulário nós preenchemos a PickList da coluna 2 da Grid1 com os valores do campo Descricao de TabFunc. Fazemos ButtonStyle=cbsAuto. OBS : O aluno agora já tem condições de entender o código do evento OnShow de Form2, que é o seguinte :

```
with Dados.Table4 do  
begin  
Dados.Table4.Open;  
While not EOF do  
begin  
DBGrid1.Columns[2].PickList.Add(FieldByName('Descricao').Value);  
Next;  
end;  
end;
```

Vamos agir de forma similar com os campos Sexo(Grid1) e Semana(Grid2). Neste caso preencheremos a PickList diretamente no Editor próprio. (Masculino;Feminino no primeiro caso e 1;2;3;4;5 no segundo caso).

- Vamos submeter nosso programa a alguns testes de verificação :
- Vamos tentar incluir uma matrícula já existente. Verificamos que o programa permite cadastrar todos os campos mas na hora de gravar o registro ele solta uma mensagem de Key Violation e não grava o registro. Isto é o que nós queremos que ocorra mas há um inconveniente, a mensagem de erro só ocorre depois que o usuário digitou todos os campos do registro, numa tabela com muitos campos isto pode se tornar irritante. Veremos posteriormente técnicas para fazer a crítica a nível de campo e não de registro. Vamos, igualmente, tentar incluir um conjunto Matrícula+Semana repetido na Grid2. O programa emite a mensagem mas, se não for possível editar, ele gravará o registro, obrigando-nos a deletá-lo posteriormente. Isto ocorre porque não temos possibilidade de Cancelar a nível da Grid2 somente. Vamos corrigir inserindo um Navigator para a Table2. Para tal façamos o Align de Grid1 = alTop e o de Panel3 de Grid2 = alBottom. Aumentemos a altura do formulário (puxando pela borda inferior) abrindo espaço para inserir outro Panel (Faça alTop) e outro Navigator(copie o de cima e mude p/DataSource2). Poderíamos, usar um navigator só e mudar a tabela apontada por programa. Isto seria feito verificando qual grade tem o foco e reapontando o navigator. O exemplo do Delphi \..\Delphi4\Demos\Db\Mastapp usa este enfoque, me parece, bem mais elegante.
- O formulário pode ser visualizado como texto pelo aluno. Clique com o botão direito do mouse sobre o formulário e escolha, no menu pop-up que se apresenta a opção ViewAsText. Examine os componentes e suas propriedades. Para voltar ao formulário, abra de novo o menu pop-up e escolha ViewAsForm. Utilizando esta opção pode-se fazer alterações em componentes e propriedades. Pode-se Copiar/Colar componentes no mesmo formulário ou de um formulário para outro. Utilizando Ctrl+C /Ctl+V (Ctrl+X/Ctrl+V se se quiser Cortar). Isto pode ser feito no formulário ou em sua tradução AsText, pode-se copiar até para o módulo .pas. O aluno só deve utilizar este artifício se estiver seguro do que faz pois pode cometer erros que acarretem a perda do formulário. Um exemplo interessante consiste em copiar para a área de transferência o texto (ViewAsText de Form2) entre **object** e **end** do objeto BitBtn1. Volte ao formulário (ViewAsForm), delete o BitBtn1 e cole o texto no formulário. O BitBtn reaparecerá.

CÁLCULO DA FOLHA DE PAGAMENTO

Prosseguindo com o nosso projeto (Pessoal), devemos preparar as rotinas de final de mês quando se calcula e se imprime a folha de pagamento. Faremos primeiro uma rotina para o cálculo da Folha. Esta rotina deverá, para cada funcionário, totalizar as Horas_na_Semana(TabDados) no campo Horas_no_mes (TabFun); obtido o total, este será multiplicado pelo salário hora para se obter o total a pagar ao funcionário. Vale lembrar que estamos fazendo apenas um exercício e não levamos em conta as várias possibilidades de uma folha de pagamento real. Vamos criar um menu principal com o nome Folha contendo dois submenus : Cálculo e Impressão. A rotina de cálculo é feita no módulo de dados e deve ser verificada pelo aluno. Tem-se dois loops de leitura sequencial de tabelas: o primeiro age sobre a tabela Table1 varrendo-a desde o primeiro até o último registro. Para cada registro lido em Tabfun, o programa varre a Table2 para totalizar as horas_na_semana. Note que First para Table2 é o primeiro registro cuja matrícula coincide com a de Tabfun. Igualmente o EOF de Table2 será encontrado ao ocorrer o primeiro registro cuja matrícula é diferente da de Table1. Ao iniciar a procedure o programa muda o curso do mouse para mostrar a ampulheta (HourGlass), indicativa de operação demorada. Ao terminar o cursor é restaurado ao Default (No nosso caso o processamento é tão rápido que não se vê a forma de ampulheta). Para se certificar de que o cursor é modificado, o aluno pode comentar (colocar entre {}) o comando que restaura o cursor ao fim da rotina. Refazer o comando depois da verificação.

Para a rotina de impressão vamos usar o QuickReportWizard (New/Business/QuickReporWizard). É um assistente bem fraquinho mas nos permite começar a listagem. Escolha o alias Pessoal; tabela TabFun, todos os campos menos sexo e idade e 'Folha de Pagamento' para título do relatório. Peça PreviewReport e em seguida Finish. A tela do QR será exibida para posterior complementação. Faça Table1.Active = true; Baixe dois componentes da paleta QReport, TQRExp e TQRLabel. Faça TQRLabel.Caption = Salário no Mes. Para TQRExp clique em Expression e, seguindo o auxiliar faça : DoubleClique Horas_no_Mes, Clique sinal , DoubleClique Salario_Hora. Clique OK. Faça Mask=#,##0.0 e Alignment = taRightJustify. Baixe um componente QRBand, faça BantType = rbSummary. Nesta banda baixe um QRLabel com capiton = "Total da Folha" e um QRExp com a expressão "SUM(Horas_no_mes*Salario_hora)".Pode-se

ver um Preview da listagem clicando com o Rightbutton domouse no formulário (fora do QReport) e selecionando Preview no menu pop-up.No formulário principal faça a chamada de menu = Form4.QuickRep1.Preview; e {Form4.QuickRep1.Print;}. A segunda opção (comentada {}) manda a listagem para a impressora. A primeira nos permite visualizar a listagem. Mande compilar e rodar.

PROJETO DE SISTEMA II

Nosso projeto terá por finalidade gerenciar uma loja de materiais de Informática.

Geraremos um programa executável com o nome de 'Lojainfo.exe' que nos permitirá :

- Criar e manter a tabela *Estoques* contendo as quantidades e valores de todos os itens mantidos em estoque, para venda.

- Criar e manter a tabela *Clientes* contendo dados cadastrais dos clientes. Esta tabela nos permitiria, entre outras coisas, o envio de mala-direta promocional, controle de crédito, etc...

- Criar e manter a tabela *Vendas* contendo a descrição dos produtos vendidos, as quantidades e os valores envolvidos e a identificação do cliente. Esta tabela nos permitiria atualizar os estoques na saída das mercadorias pela dedução das quantidades vendidas, analisar resultados pela comparação dos valores de custo e dos valores de venda, fazer análises estatísticas do tipo 'mercadoria mais vendida', 'sazonalidade de vendas', etc...

- Não o faremos pois a finalidade é apenas didática mas, se acrescentássemos tabelas de compras e de fornecedores, teríamos possibilidade de um controle completo de estoque e de contas a pagar de mercadorias, o que nos permitiria até análises financeiras mais abrangentes. Poderíamos ter também um arquivo de configuração no estilo do Windows ('lojainfo.ini'), contendo valores parametrizados tais como '% de lucro desejado', 'tabelas de desconto', etc... Este tipo de arquivo nos permite modificar valores usados pelo aplicativo sem a necessidade de modificar programas, para atender as diferentes necessidades de diferentes usuários. Uma alternativa ao arquivo .ini seria utilizar o próprio Registro do Windows que, aliás, é o recomendado pela MS...

- Algumas das funções mostradas no projeto não serão implementadas por desnecessário. Quando selecionadas estas funções, o programa deverá exibir mensagem indicando este fato.

CRIAÇÃO DO BANCO DE DADOS

Tabela '*Estoques*' :

- Entre no *Database Desktop* e escolha no menu *File*, a opção *New*, sub-opção *Table*.

- No quadro *Table Type* escolha tabela do tipo *Dbase for Windows* e, no quadro *Create Table*, crie os seguintes campos :

OBS: Caso precise deletar algum campo, use Ctrl+Del.

| Field Roster: | | | | |
|---------------|------------|------|------|-----|
| | Field Name | Type | Size | Dec |
| 1 | DESCRIÇÃO | C | 36 | |
| 2 | PR_COMPRA | N | 10 | 2 |
| 3 | QUANTIDADE | N | 5 | |
| 4 | CODIGO | N | 4 | |

- Clique no botão *Define* para criar um índice; clique sobre o campo *Codigo* no quadro *Define Index* e dê o nome *indCod* para o índice.

- Repita a operação para o campo *Descrição* e dê a este novo índice o nome *indDesc*.

- Salve a tabela com o nome *Estoques* clicando no botão *Save As*.

A tabela *Estoques* foi criada nos passos anteriores, mas está vazia. Vamos colocar alguns dados nela :

- No menu *File*, opção *Open*, sub-opção *Table*, abra a tabela *Estoques*; em seguida, no menu *Table*, escolha a opção *Edit Data*, para trabalhar no modo de edição.

- Cadastre os seguintes itens :

| Codigo | Descrição | Quantidade | Pr_Compra |
|--------|--------------------------|------------|-----------|
| 1 | PC IBM Aptiva mod. 2 | 5 | 2998,00 |
| 2 | PC Compaq mod. 1 | 10 | 2450,00 |
| 3 | Modem US-Robotics 28800 | 3 | 120,00 |
| 4 | Kit MultMedia Adventures | 22 | 240,00 |
| 5 | Scanner Genius mod. 12 | 10 | 90,00 |

- Salve os dados deslocando-se para o próximo registro em branco.

Tabela '*Clientes*' :

-Crie esta tabela utilizando passos semelhantes ao da tabela anterior.

-Campos da tabela *Clientes* :

Field Roster:

| | Field Name | Type | Size | Dec |
|---|------------|------|------|-----|
| 1 | NOME | C | 36 | |
| 2 | SEXO | C | 1 | |
| 3 | RENDA_MENS | N | 10 | 2 |
| 4 | VOL_COMPRA | N | 10 | 2 |
| 5 | ENDEREÇO | C | 36 | |
| 6 | CEP | C | 9 | |
| 7 | TELEFONE | C | 8 | |
| 8 | CODIGO | N | 4 | 0 |
| 9 | IMAGE | B | | |

- Crie dois índices : indCod com o campo *Codigo* e indNome com o campo *Nome*.

- Digite os seguintes dados nesta tabela :

| Codigo | Nome | Sexo | Renda Mensal | Volume de Compras | Endereço | CEP | Telefone |
|--------|---------------|------|--------------|-------------------|-----------------------------|-----------|----------|
| 1 | José Maria | M | 2400,00 | 500,00 | Ladeira do Escorrega, 25 | 22225-120 | 222-2222 |
| 2 | Maria José | F | 13000,00 | 22000 | Rua do Esconde-Esconde, 212 | 11111-111 | 111-1111 |
| 3 | Luiz Cláudio | M | 900,00 | 1200,00 | Travessa do X, 12 | 20001-001 | 123-4567 |
| 4 | Claudia Luz | F | 1200,00 | 5400,00 | Rua do Poste, 23 | 22225-120 | 222-2222 |
| 5 | Marcos Fortes | M | 600,00 | 100 | Av. dos Tarzans, 24 | 22225-120 | 111-1111 |

Tabela '*Vendas*' :

-Crie esta tabela utilizando passos semelhantes ao da tabela anterior.

-Campos da tabela *Vendas* :

Field Roster:

| | Field Name | Type | Size | Dec |
|---|------------|------|------|-----|
| 1 | DATA | D | | |
| 2 | COD_MERCAD | N | 4 | |
| 3 | COD_CLIENT | N | 4 | |
| 4 | QUANTIDADE | N | 4 | |
| 5 | PREÇO_UNIT | N | 10 | 2 |

- Crie o índice indData com o campo DATA.
- Digite os seguintes dados nesta tabela :

| Data | Cod_Mercadoria | Cod_Cliente | Quantidade | Preço_Unitário |
|----------|----------------|-------------|------------|----------------|
| 12.06.96 | 2 | 1 | 1 | 2450,00 |
| 12.06.96 | 3 | 1 | 1 | 120,00 |
| 12.06.96 | 5 | 4 | 3 | 90,00 |
| 13.06.96 | 4 | 4 | 12 | 210,00 |
| 14.06.96 | 4 | 1 | 3 | 240,00 |

Feche o *Database Desktop* com a opção *Exit* do menu *File*.

CRIAÇÃO DO APLICATIVO

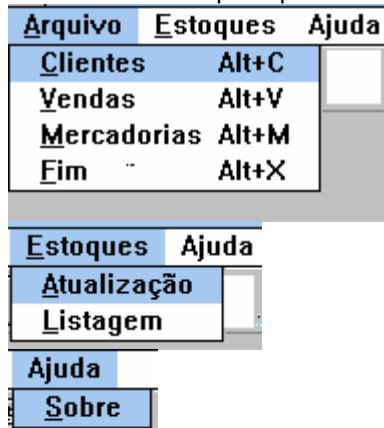
Vamos acessar o Delphi e preparar nosso aplicativo, que ficará com o visual abaixo e será precedido da seguinte Splash Screen (mostrada apenas durante o período de carga do programa) :





Passos necessários para a montagem :

- Crie um projeto a partir de um formulário em branco; dê ao formulário o nome *FormaPrincipal*, mude o caption para *Gerente de Loja* e salve o código como *frmprj* e o projeto como *lojainfo*, no diretório *C:\CDelphi\Ex05*.
- Insira um componente *Image* (da paleta *Additional*) e selecione para ele (na propriedade *Picture*) o arquivo *C:\..\Delphi3\images\splash\16color\skyline.bmp*, coloque duas legendas, um acima e outra abaixo da *Picture*. Utilize para o caption das legendas a fonte *Poster Bodonni ATT-Bold-10*
- Insira um menu principal com submenus e teclas de atalho, conforme abaixo :



- Insira um componente *Panel* apague seu caption e faça as propriedades *Align = alTop*; *Alignment=taCenter*; *BeverlInner = bvINone* e *BevelOuter = bvRaised*.
- Insira no painel recém-criado alguns *speedbuttons* com *Hints* e propriedade *ShowHint = True*. Os arquivos *.bmp* para a propriedade *Gliph* dos *speedbuttons* podem ser obtidos no diretório *C:\..\Delphi3\Images\Buttons*. A escolha pode ser algo arbitrária, pois a finalidade é apenas

ilustrativa. Verifique os arquivos *calculat.*, *comppc1*, *arrow1d*, *dooropen*, *fcabshut*, *printer* e *help*, todos *.bmp*. Note que na propriedade *Hint* dos *speedbuttons* pode-se inserir dois textos separados pelo símbolo '|'. O primeiro aparecerá junto ao *speedbutton* e o segundo aparecerá na linha de status no rodapé.

O seguintes hints devem ser inseridos nos *speedbuttons* :

- Clientes|Cadastra/Edita Clientes no SpeedButton1;
- Vendas|Movimento de Vendas no SpeedButton2;
- Mercadorias|Cadastra/Edita Mercadorias no SpeedButton3;
- Fim|Encerra o Aplicativo no SpeedButton4;
- Atualização|Atualiza Estoques de Mercadorias no SpeedButton5;
- Listagem|Lista Estoques de Mercadorias no SpeedButton6;
- Sobre|Informações de Autoria no SpeedButton7

- Para criar o rodapé, insira um componente *Statusbar* faça a propriedade *Align = alBottom* e *Height* para 25. Mude seu nome para *StatusLine*.

- Para que o texto apareça no rodapé são necessárias algumas linhas de código : Crie uma procedure *ShowHint* dentro da classe *TFormaPrincipal*, e utilize-a ao criar a forma *FormaPrincipal*. Na realidade, modifica-se a propriedade *SimpleText* da *StatusBar* para conter o texto desejado.

OBS : Se tivéssemos utilizado o *Application Wizard* na montagem, estes passos seriam feitos automaticamente pelo Delphi.

- Clique no item *Clientes* do Menu *Arquivo* e observe que o Delphi vai inserir o evento *Click1* no *Events* do submenu e vai abrir a procedure *TForm1.Clientes1Click* (no *Code Editor*) para nos permitir inserir o manipulador de eventos apropriado.

- Vá agora ao *Object Inspector* e selecione o *SpeedButton1* (que é o equivalente a opção de menu *Clientes*), clique na sua página *Events* e (no evento *OnClick*) selecione para este evento o mesmo manipulador (*Cliente1Click*) do item de menu (já que ambos perfazem a mesma função).

- Insira no corpo da *Clientes1Click* a *MessageBox* indicando *Função não implementada*. É necessário colocar alguma coisa no corpo da procedure pois, ao salvarmos o projeto, a função, se estiver vazia, será excluída.

- Repita, para os demais itens de menu, um procedimento similar ao anterior.

- Copie (através do *Clipboard*) a parte do *Hint* seguinte ao símbolo '|' dos componentes *SpeedButton* para a propriedade *Hint* dos elementos de menu correspondentes.

- Insira teclas aceleradoras nos sub-itens do menu *Arquivo* entrando na propriedade *short-cut* e selecionando o conjunto de teclas desejadas.

- Neste ponto do projeto ainda não desenvolvemos nenhuma das rotinas do menu. Por razões de organização do nosso trabalho (e para não perder os esboços de procedures abertas pelo Delphi), inserimos as mensagens de advertência abaixo indicada em cada uma das chamadas de menu.

Código da mensagem :

```
Application.MessageBox('Função não Implementada', 'A T E N Ç Ã O', mb_OK + mb_DefButton1);
```

Vamos, a partir de agora, começar a retirar as mensagens de advertência e inserir nossas funções de processamento:

Opção de menu *Fim* (no menu principal *Arquivo*):

Na procedure *TForm1.Fim1Click* substitua a *MessageBox* por *Close*;

Note que você tem várias opções para encerrar o programa :

- Clicando o item de menu *Fim* ;
- Usando as teclas aceleradoras *Alt+X* ;
- Usando o *speedbutton* correspondente :

- Usando as teclas aceleradoras de sistema (do Windows) Alt+F4;
- Abrindo o menu de sistema da janela e clicando em *Fechar*.
- Dando dois cliques sobre o menu de sistema.
- Dando um clique simples no botão Close da janela.

Opção de menu *Sobre* (no menu principal *Ajuda*) :

Vamos criar uma caixa de dialogo *About* selecionando *New/Forms/AboutBos* no menu *File*. Mude o caption para *Sobre o Aplicativo*. Mude a propriedade *Name* para *FormaSobre*, renomeie a unidade de código para *Frmsobre* e inclua *Frmsobre* na cláusula *uses* de *Frmpr*.

Na procedure *TFormaPrincipal.Sobre1Click* substitua a *MessageBox* por *'FormaSobre.ShowModal*. As caixas *About* permitem informar dados de propriedade, autores, versão e outros; para tal, modifique os labels existentes e insira outros se necessário e faça as modificações devidas para obter o visual abaixo:



CRIAÇÃO DO FORMULÁRIO CADASTRO DE CLIENTES

Aproveitamos, para este formulário, o layout do programa *Animals* contido no pacote do *Delphi1..*

O processo de pirataria do programa *Animals* consiste em copiar os arquivos *beastwin.dfm* e *beastwin.pas* para o nosso diretório de trabalho. Para incluir o novo arquivo em nosso projeto devemos utilizar *View/Project Manager/Add* e selecionar o arquivo *beastwin.pas*. Em seguida devemos mudar o nome do nosso arquivo para *FrmCad*, abrindo-o e salvando-o com novo nome em *File|Save File As...* Devemos deletar, em nosso diretório os arquivos *beastwin.pas* e *beastwin.dfm*.

Com o nosso novo formulário (recém pirateado) selecionado, vamos mudar sua propriedade *Name* para *FormaCadVis* e o *Caption* para *Gerente de Loja*. Vamos selecionar *Table1* e modificar as propriedades: *DatabaseName* para (deixar em branco); *TableName* para *Clientes* e *Active* para *True*. Vamos também, com *Table1* selecionada, dar um duplo clique sobre a mesma e remover todos os campos (no *Fields Editor*)...

Vamos deletar os componentes *DBEdit1*, *DBEdit2*, *DBEdit3* e *DBEdit4* e os labels correspondentes.

Vamos alterar o caption do componente *Panel1* para *Cadastro de Clientes*.

Vamos selecionar os componentes *DBImage1* e trocar a sua propriedade *DataField* para *Image*.

Vamos, em seguida, inserir componentes dos tipos *Label* e *TDEdit* para montar a nossa tela como mostrado na figura. Para cada componente *TDEdit* torne a propriedade *Datasource* = a *DataSource1*, em seguida selecione em *DataField* o campo correspondente. Por exemplo: No *DBEdit1*, selecione o campo *Nome*...

Alem de cadastrarmos os campos da tabela *Clientes*, podemos acrescentar uma imagem gráfica associada a cada um dos registros da tabela. Essa imagem deverá ser copiada através da *Área de Transferência*. e poderá ser a imagem editada de um arquivo gráfico.

Para transferir algumas imagens para a tabela proceda da seguinte forma :

Execute, através do WindowsExplorer o programa *clientes.exe* Para selecionar uma imagem basta clicar com o mouse no interior da mesma. Copie para o clipboard com Ctrl+C. Volte para o lojainfo, execute o programa, selecione o registro desejado e tecla Ctrl+V. A imagem será transferida para a nossa tabela. Repita este procedimento quantas vezes for necessário.

Nota : A cópia de imagens gráficas para a área de transferência e, desta para a tabela pode ser feita por programa, utilizando comandos e componente próprios. Uma consulta ao *Help* do *Delphi*, tópico *Clipboard* pode ser bastante elucidativa. O programa *clientes.exe* foi adaptado de um demo do *Delphi1* (stocks).

Pode-se também transferir direto de arquivos de imagens através do comando :
DBImage.Picture.LoadFromFile(FileName);

The screenshot shows a Windows-style application window titled "Gerente da Loja" with a sub-header "Cadastro de Clientes". The form contains the following data:

| | | |
|----------------|------------------|----------|
| Nome | Claudia Luz | |
| Codigo/Sexo | 4 | F |
| Renda Mensal | 1200 | |
| Vol de Compras | 5400 | |
| Endereço | Rua do Poste, 23 | |
| CEP/Telefone | 22225-120 | 222-2222 |

At the bottom of the form, there is a set of navigation buttons (back, forward, home, refresh, etc.) and two action buttons: "Query" and "Sair". A photograph of a woman in a red shirt is displayed in a frame on the right side of the form.

CRIAÇÃO DO FORMULÁRIO PARA QUERY DE CLIENTES

Este formulário, além de também permiti cadastrar os clientes da loja , vai nos permitir demonstrar alguns usos de SQL e também o uso do método *Print* da classe TForm.

| NOME | SEXO | RENDA_MENS | VOL_COMPRA | ENDEREÇO | CEP | TELEFONE | CODIGO |
|------------------------|------|------------|------------|-----------------------------|-----------|----------|--------|
| José Maria | M | 2400 | 600 | Ladeira do Escorrega, 25 | 22245-120 | 222-2222 | 1 |
| Maria José | F | 13000 | 22000 | Rua do Esconde-Esconde, 212 | 11111-111 | 111-1111 | 2 |
| Luiz Claudio | M | 900 | 1200 | Travessa do X, 12 | 20001-001 | 123-4567 | 3 |
| Claudia Luz | F | 1200 | 5400 | Rua do Poste, 23 | 22225-120 | 222-2222 | 4 |
| Marcia Fortes | F | 600 | 100 | Rua dos Tarzans, 24 | 22225-120 | 111-1111 | 5 |
| lotio Linhares | M | 1000 | 100 | Rua do Aquario, 22 - Lagoa | 00000-000 | 000-0000 | 6 |
| Maria Serpentina | F | 3000 | 1000 | Rua dos Ofídios, 21 | 22222-222 | 111-1111 | 7 |
| Marina Felina de Souza | F | 3000 | 2000 | Rua do Jardim Zoológico, 14 | 22222-222 | 111-1111 | 8 |
| Felicio Crista | M | 6000 | 7000 | Rua do Poleiro, 11 | 22222-222 | 111-1111 | 9 |
| Josè de Aguas Fiasas | M | 200 | 100 | Rua da Lagoinha, 12 | 22222-222 | 111-1111 | 10 |
| Undecimo de Souza | M | 11000 | 1100 | Rua da Amizade, 44 | 11111-111 | 222-2222 | 11 |
| Paulo Testa | M | 1000 | 1000 | Rua do PC, 44 | 22222-222 | 111-1111 | 12 |

- Selecione o *Form Wizard* do menu *Database*. No quadro *Form Options* escolha o botão *Create a Simple Form*. No quadro *DataSetOptions*, escolha *Create a form using TQuery objects* e clique *Next*; Escolha a tabela *Clientes.dbf* como base para o nosso formulário; selecione os campos que você usará. Clique ' >>' para selecionar todos e, em seguida, clique em *Next*; Escolha *Grid* como a forma em que os dados serão mostrados e clique em *Next*; Desmarque a opção para formulário principal (*MainForm*), e clique no botão *Finish* para gerar o formulário.
 - Mude o *caption* para *Query(Consulta) de Clientes* e o nome para *FormaCadClientes*.
 - Salve o módulo como *FrmClien*. (com *Save File As...*).
Observe que o *Wizard* já inseriu para nós, obedecendo as nossas seleções , os seguintes componentes : *DataSource1*, *DBGrid1*, *DBNavigator*, *Panel1*, *Panel2*, *Query1* e o formulário por nós renomeado para *FormaCadClien*. O *Panel1* ocupa o topo do formulário (*Align = alTop*) , o *Panel2* ocupa o restante da área do cliente (*Align = alClient*).
 - O *Navigator* completo contém 10 botões, os seis últimos são voltados para o cadastramento e edição de dados; os quatro primeiros permitem navegar pelo banco de dados e tem o seguinte significado : o primeiro nos transporta para o primeiro registro da tabela, o segundo nos leva para o registro anterior ao atual, o terceiro para o registro seguinte e o quarto para o último registro da tabela. Vamos excluir do *Navigator* os 6 últimos botões. Para tal, com o mouse sôbre a propriedade *Visible Buttons*, dê um duplo clique e serão mostrados todos os botões com propriedade *True*; passe os 6 últimos para *False*. (Um duplo clique na propriedade *VisibleButtons* fechará de novo a lista individual. Os quatro botões restantes se alargam para ocupar o espaço deixado livre; você deve reduzi-los a um tamanho menor.
 - Insira tres componentes *Button* troque seus *captions* para *SQL1*, *SQL2* e *Sair*.
 - Insira um componente *Combobox*, um pequeno label para indicar sua finalidade.
 - Insira um botão tipo *BitBtn* com o gliph indicado e o *caption* *Imprimir*. Para o gliph selecione *..\delphi3\images\buttons\print.bmp* (ou algo similar)...
- Nota : Redimensione o formulário conforme necessário.
- Para que este formulário possa ser utilizado pela aplicação o aluno deverá incluir um botão *BitBtn* com o símbolo mostrado no formulário *FormaCadVis* (utilize *..\delphi3\images\buttons\query.bmp* para a propriedade *Gliph*); na procedure associada ao evento *OnClick* deste botão inserir o comando : *FormaCadClientes.Show* e, inserir *FrmClien* na cláusula *uses* de *FrmCad*. (Poderíamos também, ter criado uma chamada de menú diretamente para o formulário de *Query*).

Reproduzimos, em seguida, trecho do *Code Editor*, contendo a codificação necessária :

```
// Arquivo de Projeto (lojainfo.dpr)
program Lojainfo;

uses
  Forms,
  Frmpr in 'FRMPRI.PAS' {FormaPrincipal},
  Frmsobre in 'FRMSOBRE.PAS' {FormaSobre},
  Frmmerc in 'FRMMERC.PAS' {FormaCadMerc},
  Frmclien in 'FRMCLIEN.PAS' {FormaCadClientes},
  Frmsql in 'FRMSQL.PAS' {FormaTextoSQL},
  Frmcad in 'FRMCAD.PAS' {FormaCadVis},
  Frmven in 'Frmven.pas' {FormaVendas},
  Splash in 'splash.pas' {SplashForm};

{$R *.RES}

begin
  { Observe a inserção da splash screen}
  SplashForm := TSplashForm.Create(Application);
  SplashForm.Show;
  SplashForm.Update;

  Application.CreateForm(TFormaPrincipal, FormaPrincipal);
  Application.CreateForm(TFormaCadClientes, FormaCadClientes);
  Application.CreateForm(TFormaCadMerc, FormaCadMerc);
  Application.CreateForm(TFormaSobre, FormaSobre);
  Application.CreateForm(TFormaTextoSQL, FormaTextoSQL);
  Application.CreateForm(TFormaCadVis, FormaCadVis);
  Application.CreateForm(TFormaVendas, FormaVendas);
  Application.CreateForm(TSplashForm, SplashForm);

  { Observe a retirada e destruição da splash screen}
  SplashForm.Hide;
  SplashForm.Free;

  Application.Run;
end.

//Unidades do projeto (.pas)
  unit Frmclien;

  interface

  uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
    StdCtrls, Forms, DBCtrls, DB, DBGrids, DBTables, Grids, ExtCtrls, Dialogs,
    Buttons, Frmsql;

  type
    TFormaCadClientes = class(TForm)
```

```
DBGrid1: TDBGrid;
Panel1: TPanel;
DataSource1: TDataSource;
Panel2: TPanel;
Query1: TQuery;
Button2: TButton;
Button3: TButton;
OpenDialog1: TOpenDialog;
ButPrint: TBitBtn;
ComboBox1: TComboBox;
DBNavigator: TDBNavigator;
Button1: TButton;
Label1: TLabel;
Label2: TLabel;
procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure ButPrintClick(Sender: TObject);
procedure ComboBox1Change(Sender: TObject);
private
  { private declarations }
public
  { public declarations }
end;

var
  FormaCadClientes: TFormaCadClientes;

implementation

  {$R *.DFM}

  procedure TFormaCadClientes.FormCreate(Sender: TObject);
  begin
    Query1.Open;
  end;

  procedure TFormaCadClientes.Button2Click(Sender: TObject);
  begin
    Close;
  end;

  procedure TFormaCadClientes.Button1Click(Sender: TObject);
  begin
    FormaTextoSQL.Show;
  end;

  procedure TFormaCadClientes.Button3Click(Sender: TObject);
  begin
    if OpenDialog1.Execute = True then
      begin
        With Query1 do
          begin
            Close;

```

```
        SQL.LoadFromFile(OpenDialog1.FileName);
        Open;
    end;
end;
end;

procedure TFormaCadClientes.ButPrintClick(Sender: TObject);
begin
    Print;
end;

procedure TFormaCadClientes.ComboBox1Change(Sender: TObject);
begin
    case ComboBox1.ItemIndex of
        0 : OpenDialog1.FileName := 'sqltex1.txt';
        1 : OpenDialog1.FileName := 'sqltex2.txt';
        2 : OpenDialog1.FileName := 'sqltex3.txt';
        3 : OpenDialog1.FileName := 'sqltex4.txt';
        4 : OpenDialog1.FileName := 'sqltex5.txt';
        5 : OpenDialog1.FileName := 'sqltex6.txt';
        6 : OpenDialog1.FileName := 'sqltex7.txt';
        7 : OpenDialog1.FileName := 'sqltex8.txt';
    end;

    With Query1 do
        begin
            Close;
            SQL.LoadFromFile(OpenDialog1.FileName);
            Open;
        end;
    end;
end.
```

O botão SQL1 habilita a abertura de vários arquivos .txt (poderíamos usar qualquer extensão) usados para acessar o banco de dados de diversas formas diferentes. Experimente com os varios arquivos para você sentir a força da linguagem SQL. Apesar do título ser *Cadastro de Clientes* você pode acessar qualquer tabela, não só a tabela Clientes. É evidente que, se escolhida outra tabela, o caption de nosso formulário não mais estará representando a realidade, pois ele é específico para a tabela *Clientes*. O aluno não terá dificuldade alguma em antever um software mais genérico, destinado a mostrar qualquer tabela; bastará mudar (em run-time) o caption do formulário para refletir o nome da tabela selecionada.

Dá-se, a seguir os textos contidos nos arquivos utilizados no exemplo :

```
sqltex1.txt :
Select * from clientes
order by codigo
```

```
sqltex2.txt:
Select * from clientes
order by nome
```

```
sqltex3.txt:
Select * from clientes
where Sexo = "M"
order by Nome
```

```
sqltex4.txt:
Select * from clientes
where Sexo = "F"
order by Nome
```

```
sqltex5.txt :
Select * from clientes
where renda_mens > 1500 and vol_compra > 10000
order by Nome
```

```
sqltex6.txt:
Select * from clientes
where Nome like "M%"
order by Nome
```

```
sqltex7.txt:
Select * from estoques
order by codigo
```

```
sqltex8.txt:
Select * from vendas
order by data
```

O botão SQL2 permite que o usuário digite uma 'query' diretamente em uma caixa de memorando e a 'query' é executada em seguida. Utilizando o *Notepad* ou outro editor de texto, visualize os arquivos utilizados em SQL1 para treinar o uso da linguagem SQL... Esta é uma excelente opção para o aluno se exercitar no uso de queries em SQL... Vale a pena consultar, no *Help* do Delphi, os tópicos referentes aSQL.

A opção SQL2 utiliza uma caixa de diálogo para a didgitção de queries. Selecione *New Dialogs* e escolha *Standard Dialog*. Mude o caption p/ *Texto de SQL*; altere o nome para *FormaTextoSQL*; renomeie o módulo para *Frmsql* e inclua-o na cláusula **uses** do módulo *FrmClien*; Baixe um componente *Tmemo* e faça-o ocupar todo o espaço de *Tbevel*. e insira na propriedade *Lines* do *Tmemo* as seguintes linhas :

```
select * from clientes
order by codigo
```

Dá-se abaixo, o código necessário a *Frmsql*.

```
unit Frmsql;
```

```
interface
```

```
uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, Buttons,
StdCtrls, ExtCtrls;
```

```
type
```

```
TFormaTextoSQL = class(TForm)
  OKBtn: TBitBtn;
  CancelBtn: TBitBtn;
  Bevel1: TBevel;
  Memo1: TMemo;
  procedure OKBtnClick(Sender: TObject);
  procedure CancelBtnClick(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
public
```



```
    { Public declarations }
end;

var
  FormaTextoSQL : TFormaTextoSQL;
implementation
uses Frmcliem;

{$R *.DFM}

procedure TFormaTextoSQL.OKBtnClick(Sender: TObject);
var
  i : integer;
begin
  Close;
  {Executa a query}
  With FormaCadClientes.Query1 do
    begin
      Close;
      SQL.Clear;
      for i := 0 to FormaTextoSQL.Memo1.Lines.Count-1 do
        SQL.Add(Memo1.Lines[i]);
      Open;
    end;
  end;

procedure TFormaTextoSQL.CancelBtnClick(Sender: TObject);
begin
  Close;
end;

end.
```

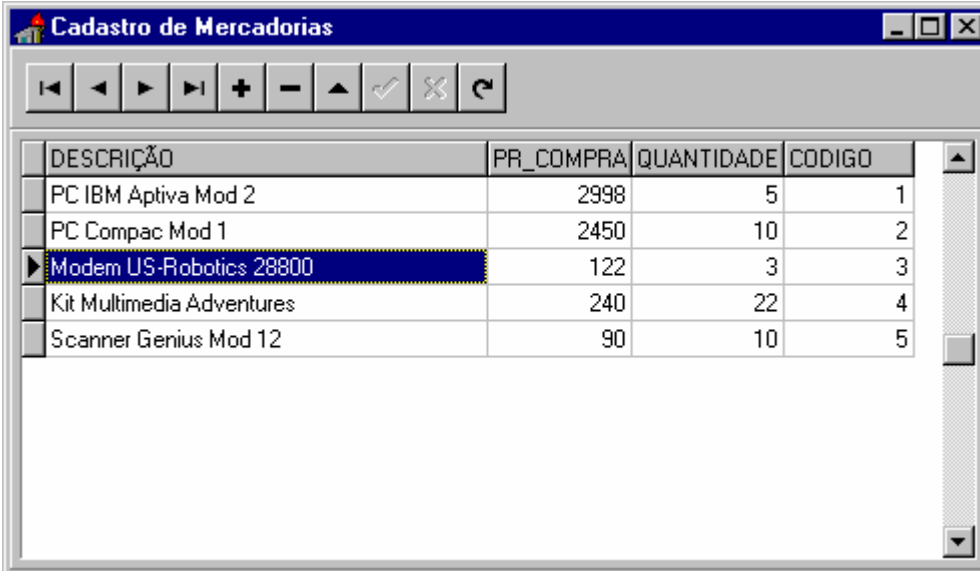
A caixa Combo SQL3 permite escolher os mesmos arquivos da SQL1 mas, desta feita, selecionando-os em uma lista explicativa.

O botão *Imprimir* utiliza o método *Print (de TForm)* para imprimir o formulário. É uma forma simples e limitada de imprimir os dados de uma tabela visíveis na janela. Maximize o formulário para obter a maior quantidade de dados possível.

A esquerda do formulário, aparece o *DBNavigator* que serve para o usuário navegar na tabela. Neste caso particular, devido a premência de espaço, deixamos visíveis apenas os elementos necessários à navegação pelo banco de dados, excluindo os demais. Nada impede, no entanto, que a tabela seja editada pois tornamos RequestLive = True na Query correspondente. O aluno deve exercitar-se, utilizando o 'navigator', o mouse e o teclado.

CRIAÇÃO DO FORMULÁRIO CADASTRO DE MERCADORIAS

Este formulário vai servir para cadastrarmos as mercadorias da loja e terá o lay-out abaixo:



| DESCRIÇÃO | PR_COMPRA | QUANTIDADE | CODIGO |
|---------------------------|-----------|------------|--------|
| PC IBM Aptiva Mod 2 | 2998 | 5 | 1 |
| PC Compac Mod 1 | 2450 | 10 | 2 |
| Modem US-Robotics 28800 | 122 | 3 | 3 |
| Kit Multimedia Adventures | 240 | 22 | 4 |
| Scanner Genius Mod 12 | 90 | 10 | 5 |

Utilize o *Form Wizard* e, utilizando um procedimento semelhante ao do formulário anterior, selecione o objeto *Tquery* e a forma de apresentação com o objeto *Grid*.

Mude o *Caption* para Cadastro de Mercadorias e o *Name* para FormaCadMerc e renomeie o módulo para Frmmerc.

Faça a propriedade *Query1.Active* igual a *True* para vermos os dados do BD durante a montagem.

Antes de compilar faça a propriedade *Query1.Active = False*.

Se o Delphi abrir a query na procedure do evento *OnFormCreate*, retire-a e utilize a *FormShow* para abrir a *Query1* e *FormClose* p/ Fechar a *Query1*.

Redimensione o formulário (clcando e arrastando nas fronteiras) até que todas as colunas estejam visíveis.

Selecione a propriedade *SQL* do *Query1* para ver as strings de SQL criadas pelo Delphi e que deverão ser iguais as mostradas abaixo:

Select

```
estoques."DESCRIÇÃO",
estoques."PR_COMPRA",
estoques."QUANTIDADE",
estoques."CODIGO"
```

From estoques

Insira *Frmmerc* na clausula *uses* de *Formapri* para que haja o devido reconhecimento.

Torne *False* a propriedade *Visible* de *FormaCadMerc*.

Faça *Query1.RequestLive = True* para poder editar a query.

Note que utilizamos o *DBNavigator* completo (inserido pelo *Wizard*) já que, pretendemos poder cadastrar e editar mercadorias neste formulário. Note também que traduzimos os *Hints* para o Português.

Para fazer isto, inserimos as strings em Português na propriedade *Hints* e fizemos a propriedade *ShowHint* igual a *True*. As strings de hint devem ser separadas pelo símbolo '|'; a parte a esquerda deste símbolo será mostrada junto ao ícone do navigator e a parte a direita será mostrada na status line.

Utilizamos a tradução seguinte (ver propriedade *Hints*).

| | |
|----------------------------|--|
| Inicio Primeiro registro | --- equivalente a <i>First Record</i> |
| Anterior Registro Anterior | --- equivalente a <i>Prior Record</i> |
| Seguinte Registro Seguinte | --- equivalente a <i>Next Record</i> |
| Final Ultimo Registro | --- equivalente a <i>Last Record</i> |
| Incluir Incluir Registro | --- equivalente a <i>Insert Record</i> |

| | | |
|------------------------------|-----|------------------------------------|
| Deletar Deletar Registro | --- | equivalente a <i>Delete Record</i> |
| Editar Editar Registro | --- | equivalente a <i>Edit Record</i> |
| Gravar Gravar Alterações | --- | equivalente a <i>Post Edit</i> |
| Cancelar Cancelar Alterações | --- | equivalente a <i>Cancel Edit</i> |
| Retaurar Restaurar Dados | --- | equivalente a <i>Refresh Data</i> |

CRIAÇÃO DO FORMULÁRIO MOVIMENTO DE VENDAS

Este formulário vai servir para cadastrarmos as vendas da loja e terá o lay-out abaixo:



O formulário foi criado usando-se o *FormWizard*. Escolhemos o tipo *Master/Detail* para relacionar duas tabelas. Nosso formulário, baseado na tabela *Vendas.dbf* mostra os dados *Nome* e *Telefone* (da tabela *clientes*). A relação entre as duas tabelas foi feita usando-se o campo indexador *Codigo* (em *clientes.dbf*) e *Cod_Cliente* (em *vendas.dbf*).

Não usamos SQL neste formulário...

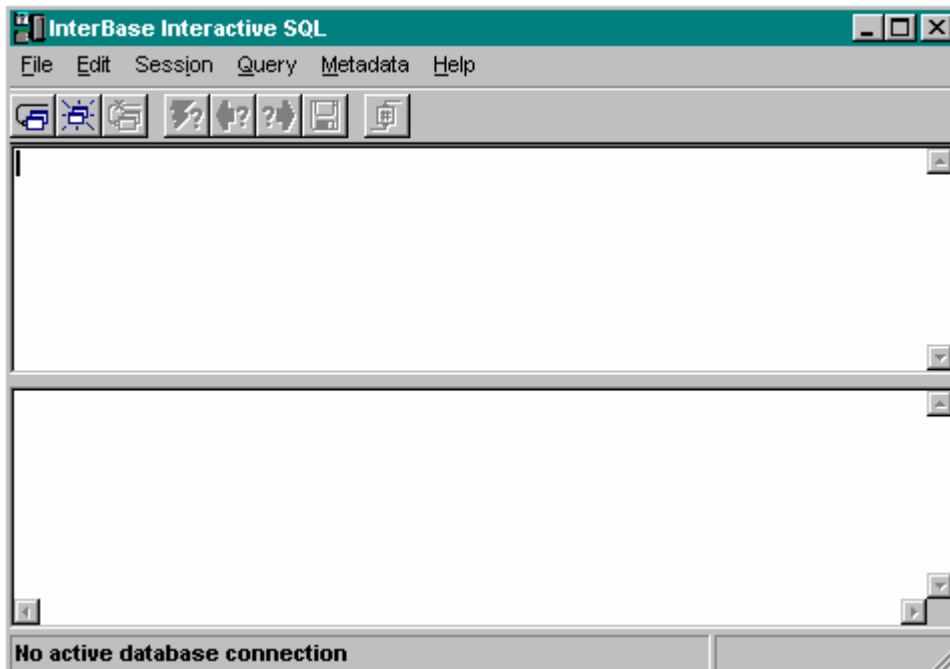
Note que utilizamos o *DBNavigator* completo (inserido pelo *FormWizard*) já que, pretendemos poder cadastrar e editar o movimento de vendas.

Colocamos também um *BitBtn* com *Kind = Close* (*Caption = &Fechar*) para fecharmos o formulário (sem precisarmos escrever código para isso).

USANDO O INTERBASE

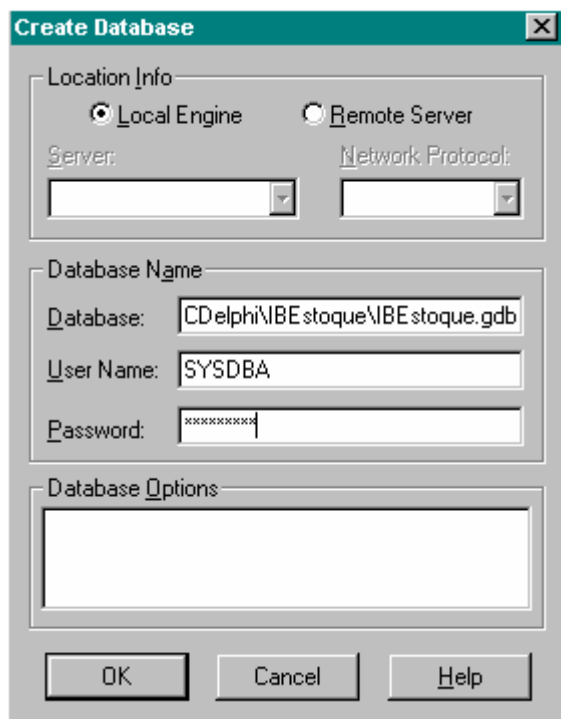
Vamos utilizar o Interbase (Versão WI-V5.1.1) em um projeto simples, destinado apenas a nos familiarizar com os aspectos iniciais deste BD. Vamos utilizar a versão para desenvolvimento que acompanha o Delphi 4.

No grupo de programas do Interbase 5.0 vamos selecionar a ferramenta W-ISQL. Esta é a ferramenta principal para os desenvolvedores, pois nos permite executar todas as operações de DDL e DML do BD.



Vamos criar um BD de nome IBEstoque.gdb. (.gdb) é a extensão padrão do Interbase. Criar o BD é a única operação que não pode ser feita via SQL.

No Menu File selecionemos 'Create Database...' e preenchamos a caixa de diálogo que se apresenta, com os dados mostrados abaixo :



OBS : SYSDBA é o User Name default do IB e a senha digitada deve ser “masterkey”.

Após o OK, se tudo ocorrer normalmente, o nome do BD aparecerá na barra de status do WISQL, indicando que a conexão com o BD foi estabelecida.

A partir deste momento podemos trabalhar com comando SQL diretamente. Estes comandos devem ser digitados na janela superior do WISQL (os resultados são mostrados na janela inferior).

Vamos, a título de aprendizado, criar uma tabela chamada TabelaBoba com os campos :

- CódigoBobo integer, NomeBobo varchar (30) e EnderecoBobo varchar(20) :

Comando SQL : Create table TabelaBoba(CódigoBobo integer, NomeBobo varchar(30),EnderecoBobo varchar(20));

Vamos utilizar o menu Metadata para verificar o nosso BD até o momento.

Vamos povoar a nossa tabela com alguns registros :

Comando SQL : Insert into TabelaBoba (CódigoBobo, NomeBobo, EnderecoBobo) values (1, "Bobo1", "End Bobo1");

Em seguida usemos o Select seguinte para verificar se a inclusão funcionou :

Comando SQL : Select * from TabelaBoba;

Vamos inserir mais um registro :

Comando SQL : Insert into TabelaBoba values (2, "Bobo2", "End Bobo2"); // Podemos fazer esta simplificação sempre que inserirmos valores para todos os campos, na ordem em que eles ocorrem na tabela.

Vamos deletar o 2º registro :

Comando SQL : Delete from TabelaBoba where CódigoBobo = 2;

Vamos refazer o Select (Query-previous) e verificar que a exclusão realmente ocorreu.

OBS : Delete sem a clausula Where DELETA TODOS OS REGISTROS DA TABELA.

Os comando utilizados para serem tornado efetivos (gravados no BD) exigem que se comande 'Commit'. Nós utilizaremos 'Rollback' pois fizemos apenas alguns testes...

Vamos dar um Drop database pois fizemos apenas alguns testes e não queremos conservar as tabelas criadas. Os comandos DDL não precisam de Commit, eles são efetivados no momento em que são inseridos.

Para facilidade de trabalho o WISQL permite o uso de arquivos de script (arquivos texto com extensão .sql contendo comandos SQL). Vamos gerar o arquivo IBEstoque.sql (no Bloco de Notas), contendo os comando abaixo :

Arquivo de script IBEstoque1

```
/* Exercicio sobre Interbase – Domains e criação de tabelas*/

/* Este arquivo cria o banco de dados IBEstoque */
CONNECT "\cdelphi\IBEstoque\IBEstoque.gdb"
USER "SYSDBA"
PASSWORD "masterkey";

/* Poderiamos ter usado CREATE ao invés de CONNECT */

/* Domain definitions */
CREATE DOMAIN TDINHEIRO AS NUMERIC(15, 2)
    DEFAULT 0
    CHECK (VALUE > 0);

CREATE DOMAIN TDESCRICAO AS VARCHAR(40);

CREATE DOMAIN TCODIGO AS INTEGER;

/* Table: ESTOQUES, Owner: SYSDBA */
CREATE TABLE ESTOQUES
(
    CODIGO TCODIGO NOT NULL,
    DESCRICAO TDESCRICAO,
    PR_COMPRA TDINHEIRO NOT NULL,
    QUANTIDADE INTEGER,
    PRIMARY KEY (CODIGO)
);

/* Table: SAIDAS, Owner: SYSDBA */
CREATE TABLE SAIDAS
(
    CODIGO TCODIGO,
    DATA DATE,
    DESCRICAO TDESCRICAO,
    QUANTIDADE INTEGER,
    PRECO_UNIT TDINHEIRO,
    FOREIGN KEY (CODIGO) REFERENCES ESTOQUES (CODIGO)
);
```

```
/* Table: ENTRADAS, Owner: SYSDBA */  
CREATE TABLE ENTRADAS
```

```
(  
  CODIGO TCODIGO,  
  DATA DATE,  
  DESCRICAO TDESCRICAO,  
  QUANTIDADE INTEGER,  
  CUSTO_TOTAL TDINHEIRO,  
  FOREIGN KEY (CODIGO) REFERENCES ESTOQUES (CODIGO)  
);
```

OBS : 1) Domains são tipos predefinidos correspondentes ao Dicionário de Dados. Conceitualmente são equivalentes ao tipos da linguagem.

2) Os conceitos de Primary Key e Foreign Key são os usuais da modelagem de dados.

Arquivo de script IBEstoque2

```
/* Exercício sobre Interbase – Criação de Triggers*/
```

```
CONNECT "\cdelphi\IBEstoque\IBEstoque.gdb"  
USER "SYSDBA"  
PASSWORD "masterkey";
```

```
SET TERM ^ ;
```

```
/* Triggers only will work for SQL triggers */
```

```
/* INCLUSÕES */
```

```
CREATE TRIGGER INCLUSAO_ENTRADAS FOR ENTRADAS  
ACTIVE AFTER INSERT POSITION 0  
AS  
BEGIN  
  UPDATE ESTOQUES  
  SET ESTOQUES.VALOR_ESTOQUE = ESTOQUES.VALOR_ESTOQUE +CUSTO_TOTAL,  
  ESTOQUES.QUANTIDADE = ESTOQUES.QUANTIDADE +NEW. QUANTIDADE  
  WHERE ESTOQUES.CODIGO = NEW.CODIGO;  
END  
^
```

```
CREATE TRIGGER INCLUSAO_SAIDAS FOR SAIDAS  
ACTIVE AFTER INSERT POSITION 0  
AS  
BEGIN  
  UPDATE ESTOQUES  
  SET ESTOQUES.VALOR_ESTOQUE = ESTOQUES.VALOR_ESTOQUE -  
  NEW.QUANTIDADE*(ESTOQUES.VALOR_ESTOQUE/ESTOQUES.QUANTIDADE),
```

```
ESTOQUES.QUANTIDADE = ESTOQUES.QUANTIDADE - NEW.QUANTIDADE
WHERE ESTOQUES.CODIGO = NEW.CODIGO;
END
^
```

```
/* EXCLUSÕES */
```

```
CREATE TRIGGER EXCLUSAO_ENTRADAS FOR ENTRADAS
ACTIVE AFTER DELETE POSITION 0
AS
BEGIN
    UPDATE ESTOQUES
    SET ESTOQUES.VALOR_ESTOQUE = ESTOQUES.VALOR_ESTOQUE - OLD.CUSTO_TOTAL,
    ESTOQUES.QUANTIDADE = ESTOQUES.QUANTIDADE - OLD.QUANTIDADE
    WHERE ESTOQUES.CODIGO = OLD.CODIGO;
END
^
```

```
CREATE TRIGGER EXCLUSAO_SAIDAS FOR SAIDAS
ACTIVE AFTER DELETE POSITION 0
AS
BEGIN
    UPDATE ESTOQUES
    SET ESTOQUES.VALOR_ESTOQUE = ESTOQUES.VALOR_ESTOQUE +
    (ESTOQUES.VALOR_ESTOQUE/ESTOQUES.QUANTIDADE)*OLD.QUANTIDADE,
    ESTOQUES.QUANTIDADE = ESTOQUES.QUANTIDADE + OLD.QUANTIDADE
    WHERE ESTOQUES.CODIGO = OLD.CODIGO;
END
^
```

```
/* UPDATES */
```

```
CREATE TRIGGER UPDATE_ENTRADAS FOR ENTRADAS
ACTIVE AFTER UPDATE POSITION 0
AS
BEGIN
    UPDATE ESTOQUES
    SET ESTOQUES.VALOR_ESTOQUE = ESTOQUES.VALOR_ESTOQUE - OLD.CUSTO_TOTAL + NEW.
    CUSTO_TOTAL,
    ESTOQUES.QUANTIDADE = ESTOQUES.QUANTIDADE + NEW.QUANTIDADE - OLD.QUANTIDADE
    WHERE ESTOQUES.CODIGO = OLD.CODIGO;
END
^
```

```
CREATE TRIGGER UPDATE_SAIDAS FOR SAIDAS
ACTIVE AFTER UPDATE POSITION 0
AS
BEGIN
    UPDATE ESTOQUES
    SET ESTOQUES.VALOR_ESTOQUE = ESTOQUES.VALOR_ESTOQUE +
    (ESTOQUES.VALOR_ESTOQUE/ESTOQUES.QUANTIDADE)*(OLD.QUANTIDADE-NEW.QUANTIDADE),
```



```
ESTOQUES.QUANTIDADE = ESTOQUES.QUANTIDADE + OLD.QUANTIDADE - NEW.QUANTIDADE
WHERE ESTOQUES.CODIGO = OLD.CODIGO;
END
^
```

OBS : 1)Triggers são comando SQL que são executados na ocorrência dos seguintes eventos do BD, Inclusão, Exclusão ou Atualização (update) de registros.

2)Os conceitos de Old e New são exclusivos de triggers.

TESTANDO O BD

Fizemos um programinha de teste (Project1). Está no diretório C:\Cdelphi\IBEstoque, onde colocamos também o BD. Para facilitar o uso criamos um Alias, utilizando o SQL Explorer (menu do Delphi Database/Explore) :

```
AliasName = IBEstoque
ServerName = C:\CDELphi\IBEstoque\IBEstoque.gdb
UserName = SYSDBA
```

Para utilizar o BD no Delphi, baixamos o componente DataBase1 e alteramos as seguintes propriedades :

```
Name = Estoque
Password = masterkey
LoginPrompt = False
```

OBS: Fizemos LoginPrompt = False para não precisarmos ficar digitando senha a todo o instante. Ao encaminhar um projeto ao usuário, o LoginPrompt deverá ser restaurado.

O aluno deve fazer vários testes incluindo, excluindo e alterando registros das tabelas Entradas e Saidas; após fazer alterações clique em Aplicar e veja o efeito das alterações na tabela Estoques.

Trata-se de um exercício e por isso não se cogitou de alguns casos que podem conduzir a erros, tais como estoque negativo, data inválida, etc...