**UNIVERSIDADE DE SÃO PAULO**
**Instituto de Ciências Matemáticas e de Computação**

ISSN - 0103-2585

# Programação Delphi apoiando a construção de aplicativos de Bases de Dados

**Enzo Seraphim**
**Renata Pontin de Mattos Fortes**

*Notas Didáticas n$^0$ 42*

São Carlos, dezembro de 1999

# Índice

# 1 INTRODUÇÃO

A disciplina de Laboratório de Base de Dados ministrada aos alunos de cursos de Computação visa proporcionar o desenvolvimento de aplicações práticas utilizando sistemas de gerenciamento de bases de dados relacionais e ferramentas de apoio. Além disso, como disciplina "optativa", ela proporciona a consolidação da teoria apresentada na disciplina de Banco de Dados que é seu pré-requisito.

Para tanto, aulas expositivas seguidas de demonstração prática em laboratório têm sido adotadas como metodologia do ensino. Os exercícios são então elaborados em aulas de laboratório assistidas.

Nesse contexto, este conjunto de notas didáticas se destina a viabilizar um suporte para a utilização de Delphi como ferramenta de apoio para as práticas necessárias de disciplinas que em seu conteúdo programático, contam com o desenvolvimento de aplicativos de Bases de Dados tendo por *front-end* a programação em Delphi. Este material foi em grande parte coletado de informações disponíveis na *World-Wide Web*.

Desde que a primeira versão do Delphi foi lançada, em 1995, tem se mostrado muito atrativo para o desenvolvimento em ambiente operacional Windows. Numa relação com outros ambientes de programação, podemos dizer que o Delphi possui muitos dos recursos poderosos do C++ e a facilidade do Visual Basic.

A principal vantagem do Delphi está na linguagem usada, Object Pascal, que é uma evolução do Pascal padrão. O Pascal surgiu no final dos anos 60 e, até hoje, é usada como uma das primeiras linguagens de programação para estudantes de computação. Em 1984, a Borland lançou o Turbo Pascal, que se firmou como um bom compilador de Pascal para o mercado e, a partir de então, passou a incluir novos recursos nesta linguagem, como Units e Objetos, até a ascensão do Windows, quando foi lançado o Turbo Pascal for Windows. Depois surgiu o Borland Pascal, cuja linguagem é considerada a primeira versão da Object Pascal. Na sua versão atual, usada pelo Delphi, a Object Pascal é uma linguagem sólida e respeitada, sem perder sua peculiar facilidade.

No Delphi, a criação de aplicativos começa com a montagem de componentes em janelas, como se fosse um programa gráfico, o usuário também pode utilizar de componentes desenvolvidos por terceiros ou criar seus próprios componentes.

O Delphi vem com todas as ferramentas necessárias para a criação de bancos de dados dBase e Paradox, além de uma versão do Interbase, permitindo a criação de aplicativos com banco de dados sem a necessidade de aquisição de outro programa. O Delphi também tem acesso a bases de dados como Foxpro, Access, Informix, SYBASE, Oracle, SQL Server e DB2, além de qualquer outro banco de dados para Windows compatível com ODBC.

Atualmente, podem ser encontradas as seguintes versões de Delphi:

- Delphi Standard → para estudantes, sem classes para armazenamento de estruturas de dados.
- Delphi Professional → com classes para armazenamento de estruturas de dados, mas sem suporte a arquitetura cliente/servidor.
- Delphi Client/Server Suite → versão completa com fontes dos componentes.

## *1.1 Principais Características do Delphi*

- **Construtor Visual de Interface com o Usuário -** o IDE (*Interface Development Environment*) permite criar visualmente aplicações Client/Server de forma rápida através da seleção de componentes na paleta.
- **Arquitetura Baseada em Componentes -** os componentes asseguram que as aplicações Delphi sejam robustas, reutilizáveis e de fácil manutenção. Com facilidade de criação de componentes nativos, além de controles ActiveX, inclusive com disponibilidade do código fonte dos componentes padrão
- **Compilador de Código Nativo de Alta Performance -** compilador/otimizador de código mais rápido do mercado, gerando executáveis rápidos e puros, sem *run-time*

- **Tow-Way Tools -** a capacidade de alternar entre um form e seu código permite aos desenvolvedores trabalhar tanto na edição de texto como no modo de design visual através de total sincronização do código fonte com a representação visual.
- **Biblioteca de Componentes Visuais -** a biblioteca de componentes visuais (VCL-*Visual Component Library*) consiste de objetos reutilizáveis incluindo objetos padrão de interface com o usuário, gerenciamento de dados, gráficos e multimidia, gerenciamento de arquivos e quadros de dialogo padrão. A Client/Server edition inclui o código fonte do Visual Component Library.
- **Arquitetura Aberta -** a arquitetura do IDE permite adicionar componentes e ferramentas personalizadas e de terceiros.
- **Linguagem Orientada a Objetos -** o Delphi utiliza o *Object Pascal*, que oferece a facilidade de programação em 4GL de alto nível com a performance e poderio de um 3GL. Fluxo de programação baseado em eventos. Suporte a manipulação de exceções, que permite criar aplicações mais robustas e com maior segurança.
- **Suporte à Tecnologia do Windows -** o Delphi é compatível com a tendência da tecnologia Windows, incluindo suporte a OLE 2.0,DDE, DCOM, VBXs e ODBC.
- **Depurador Gráfico -** o Debugger permite encontrar e eliminar "bugs" em seu código.
- **Edição Estilo Brief -** o Editor permite a utilização de um conjunto de símbolos para expressões. Consulte Brief Regular Expressions no Help on-line.
- **Ambiente Personalizável -** a opção de menu Environment Options permite personalizar seu ambiente para o máximo de produtividade.
- **Object Browser -** o Object Browser permite a visualização da hierarquia dos objetos na visual component library.
- **Gerenciador de Projetos -** o Project Manager oferece uma visualização de todos os forms e units de um determinado projeto e oferece um mecanismo conveniente para gerenciar projetos.
- **Experts -** Uma variedade de Experts o guia através do desenvolvimento de tipos padrões de forms. Por exemplo, o Database form expert auxilia-o na construção de forms que exibam dados em bancos de dados locais ou remotos.
- **Gerador de Relatórios -** o *ReportSmith*™ oferece a mais avançada ferramenta de geração de relatórios para desenvolvedores que precisem criar relatórios que acessem grandes volumes de dados, com utilização de componentes nativos.

### *1.2 Características do Delphi Client/Server*

O Delphi Client/Server Edition inclui todas as características do Delphi Professional e as seguintes características específicas ao ambiente Client/ Server:

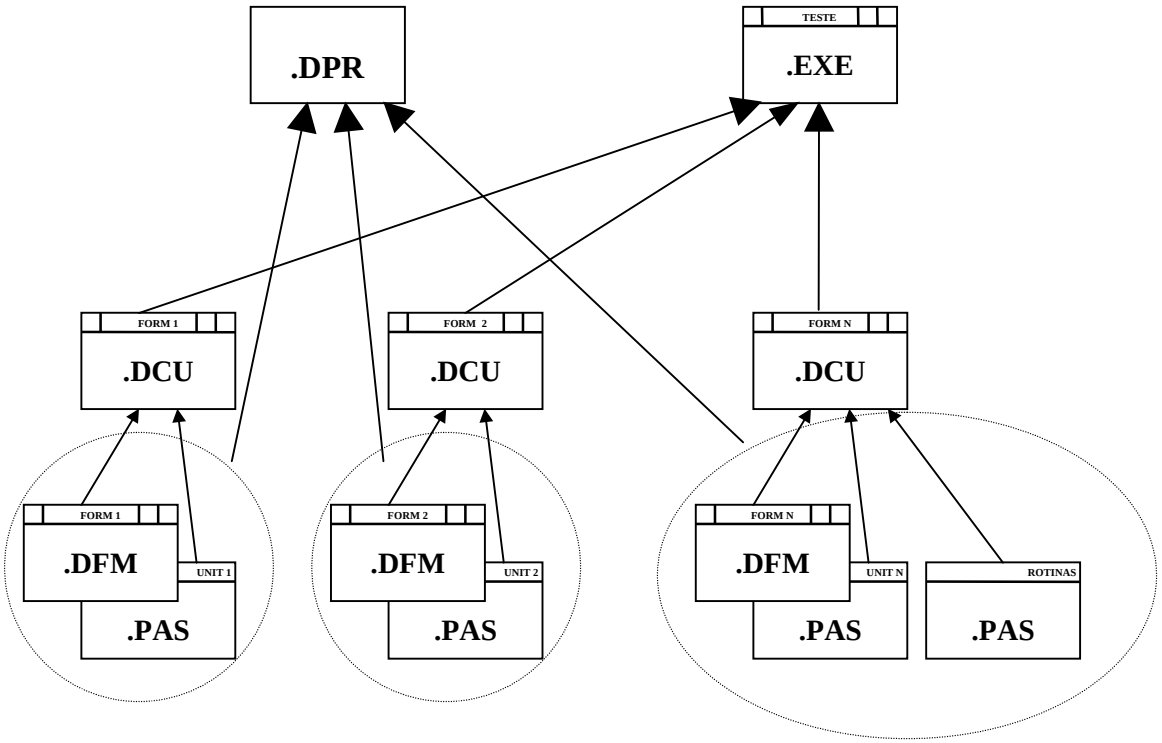
- **Conectividade de Alta Performance -** os SQL Links oferecem acesso de alta performance aos drives nativos, conectando com bancos de dados populares, incluindo Oracle, Sybase,Informix, e InterBase.
- **Suporte a Team Development -** Intersolv PVCS permite que grupos de desenvolvedores trabalhem juntos com códigos fonte integrados, check-in, check-out e gerenciamento de controle de versão.
- **Construtor Visual de Query -** o Visual Query Builder oferece uma ferramenta visual para criar facilmente queries sofisticadas e gerar automaticamente o código SQL correspondente.
- **Objetos Distribuídos -** capacidade de criação de aplicações multi-tier, com objetos distribuídos com Corba.

## 2 O AMBIENTE Delphi

### *2.1 Como é formada uma Aplicação em Delphi*

Quando o desenvolvedor abre um projeto no Delphi, já lhe é mostrada uma UNIT com várias linhas de código. Este texto tem como objetivo explicar um pouco da estrutura que ele usa.

Um projeto Delphi tem, inicialmente, duas divisões: uma **UNIT**, que é associada a um Form, e outra **Project**, que engloba todos os FORM e UNITs da aplicação. Em Delphi temos: o Project, os Forms e as Units. Para todo Form temos pelo menos uma UNIT (Código do Form), mas temos UNITs sem Form (códigos de procedures, funções, etc). A figura a seguir ilustra a formação de uma aplicação Delphi e o Quadro 1 lista as extensões dos arquivos gerados no desenvolvimento de aplicativos em Delphi.




**Quadro 1**. Arquivos Gerados no desenvolvimento em Delphi

| Extensão Arquivo | Definição | Função |
|---|---|---|
| .DPR | Arquivo do Projeto | Código fonte em Pascal do arquivo principal do projeto. Lista todos os formulários e units no projeto, e contém código de inicialização da aplicação. Criado quando o projeto é salvo. |
| .PAS | Código fonte da Unit (Object Pascal) | Um arquivo .PAS é gerado por cada formulário que o projeto contém. Seu projeto pode conter um ou mais arquivos .PAS associados com algum formulário. Contem todas as declarações e procedimentos incluindo eventos de um formulário. |
| .DFM | Arquivo gráfico do formulário | Arquivo binário que contém as propriedades do desenho de um formulário contido em um projeto. Um .DFM é gerado em companhia de um arquivo .PAS para cada formulário do projeto. |
| .OPT | Arquivo de opções do projeto | Arquivo texto que contém a situação corrente das opções do projeto. Gerado com o primeiro salvamento e atualizado em subseqüentes alterações feitas para as opções do projeto. |
| .RES | Arquivo de Recursos do Compilador | Arquivo binário que contém o ícone, mensagens da aplicação e outros recursos usados pelo projeto. |

| | | |
|---|---|---|
| .~DP | Arquivo de Backup do Projeto | Gerado quando o projeto é salvo pela segunda vez. |
| .~PA | Arquivo de Backup da Unit | Se um .PAS é alterado, este arquivo é gerado. |
| .~DF | Backup do Arquivo gráfico do formulário | Se você abrir um .DFM no editor de código e fizer alguma alteração, este arquivo é gerando quando você salva o arquivo. |
| .DSK | Situação da Área de Trabalho | Este arquivo armazena informações sobre a situação da área de trabalho especifica para o projeto em opções de ambiente (Options Environment). |

Obs.: .~DF, .~PA , .~DP são arquivos de backup( Menu Options, Enviroment, Guia Editor Display, Caixa de Grupo Display and file options, opção Create Backup Files, desativa o seu salvamento).

Devido a grande quantidade de arquivos de uma aplicação, cada projeto deve ser montado em um diretório específico.

## *2.2 Código fonte do arquivo Project(.DPR)*

Neste arquivo está descrito o código de criação da aplicação e de seus formulários. O arquivo Project tem apenas uma seção. Esta seção é formada pelo seguinte código:

PROGRAM - Define o Projeto;
USES - Cláusula que inicia uma lista de outras unidades.
Forms = É a unidade do Delphi que define a forma e os componentes do aplicativo
In = A clausula indica ao compilador onde encontrar o arquivo Unit.
Unit1 = A unidade que você criou
{$R *.RES} - Diretiva compiladora que inclui o arquivo de recursos.

Quando se abre um projeto novo, o Project possuirá o seguinte conteúdo:

```
program Project1;

uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1};

{$R *.RES}

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

## *2.3 Código fonte do arquivo Unit (.PAS)*

Nesta divisão são escritos os códigos dos seus respectivos forms (Unit1 = Form1). Assim, são definidos os códigos de cada procedimento dos componentes que você colocar no Form.

- **Seção Unit -** declara o nome da UNIT.
- **Seção Uses -** contém as UNITS acessadas por este arquivo.
- **Seção Interface -** nesta seção estão as declarações de constantes, tipos de variáveis, funções e procedures gerais da Unit/Form. As declarações desta seção são visíveis por qualquer Unit. Esta seção é formada pelo seguinte código:
  INTERFACE - palavra que inicia a seção;
  USES - cláusula que inicia uma lista de outras unidades compiladas (units) em que se basea:
  SysUtils = utilitários do sistema (strings, data/hora, gerar arquivos)

WinProcs = acesso a GDI, USER e KERNEL do Windows
Wintypes= tipos de dados e valores constantes
Messages=constantes com os números das mensagens do Windows e tipos de dados das Mensagens
Classes=elementos de baixo nível do sistema de componentes
Graphics=elementos gráficos
Controls=elementos de nível médio do sistema de componentes
Forms=componentes de forma e componentes invisíveis de aplicativos
Dialogs=componentes de diálogo comuns

- **Seção Type -** declara os tipos definidos pelo usuário. Subseções: Private, declarações privativas da Unit. Public declarações publicas da Unit.
- **Seção Var -** declara as variáveis privadas utilizadas.
- **Seção Implementation -** contém os corpos (códigos que implementam) das funções e procedures declaradas nas seções Interface e Type. Nesta seção também estão definidos todos os procedimentos dos componentes que estão incluídos no Form. As declarações desta seção são visíveis apenas por ela mesma. Esta seção é formada pelo seguinte código:
  {$R*.DFM} - Diretiva compiladora que inclui toda a interface, propriedades da forma e componentes do arquivo *.DFM
  {$S+} - Diretiva compiladora que ativa verificação de pilha.
- **Seção uses adicional -** serve para declarar Units que ativam esta.
- **Inicialization -** nesta seção, que é opcional, pode ser definido um código para proceder as tarefas de inicialização da Unit quando o programa começa. Ela consiste na palavra reservada inicialization seguida por uma ou mais declarações para serem executadas em ordem.

Quando se abre um projeto novo, a Unit possuirá o seguinte conteúdo:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{Uses Adicional}

{Initialization}

end.
```

## 2.4 Arquivos Gerados pela Compilação

| Extensão Arquivo | Definição | Função |
|---|---|---|
| .EXE | Arquivo compilado executável | Este é um arquivo executável distribuivel de sua aplicação. Este arquivo incorpora todos os arquivos .DCU gerados quando sua aplicação é compilada. O Arquivo .DCU não é necessário distribuir em sua aplicação. |
| .DCU | Código objeto da Unit | A compilação cria um arquivo .DCU para cada .PAS no projeto. |

Obs.: Estes arquivos podem ser apagados para economizar espaço em disco.

## 2.5 Ambiente de Programação

- **Janela Principal -** a janela principal do Delphi é composta pela barra de menus, barra de ferramentas e paleta de componentes. Para personalizar a barra de ferramentas, basta clicar em Properties no menu de contexto. Na paleta de componentes estão os controles usados nas aplicações, agrupados em guias por categorias.
- **Object Inspector -** no Object Inspector podemos manipular, em tempo de projeto, as propriedades e eventos dos componentes. Você também pode selecionar um componente usando o Seletor de Objetos, no topo do Object Inspector.
- **Propriedades -** são as características de um componente. Para mudar uma propriedade, selecione o componente no Form Designer ou no Object Selector, localize a propriedade, na guia Properties e mude o valor na coluna à direita. A edição de propriedades pode ser simples, por lista suspensa, caixa de dialogo ou com propriedades aninhadas.
- **Eventos -** numa definição inicial, eventos podem ser vistos como chamadas a métodos em resposta a determinadas mensagens do Windows. Para criar um método para um evento, selecione o componente e clique duas vezes na coluna à direita do evento na guia Events do Object Inspector, o Delphi faz todas as declarações necessárias e mostra o método pronto para ser programado no Editor de Código. Para que um evento chame um método já definido, em vez de clicar duas vezes na coluna à direita do evento, você deve usar a lista suspensa.
- **Form Designer -** o Form Designer é onde são desenhados os Forms das aplicações, com a inserção de componentes. No menu de contexto do Form, você pode clicar em View as Text para editar a descrição textual do Form e de seus componentes no Editor de Código, essas informações são gravadas em um arquivo binário com a extensão DFM, para voltar ao modo de exibição normal, escolha View as Form no menu de contexto do Editor de Código.
- **Manipulando Componentes**
  - **Incluir**: Selecionar o componente na paleta e clicar no Form Designer.
  - **Redimensionar**: Clicar no componente e arrastar as alças de borda. Podemos usar SHIFT+SETAS para redimensionar o componente fora da grade de alinhamento do Form.
  - **Mover**: Arrastar o componente. Podem ser usadas também operações de recortar, copiar e colar, além de CTRL+SETAS para mover o componente para fora da grade de alinhamento do Form.
  - **Selecionar**: Segurar SHIFT para selecionar vários componentes individuais e CTRL para escolher uma área retangular do Form e selecionar todos os componentes nesta área.
  - **Alinhamento**: Para alinhar componentes selecione-os e escolha View / Alignment Palette.
  - **Menu de Contexto**: Bring To Front / Send To Back, para trazer o componente para frente ou enviar para trás, Tab Order para mudar a ordem de tabulação, além de Align To Grid, para alinhar os componentes selecionados à Grade do Form Designer.
- **Editor de Código -** para escrever o código, usamos o Editor de Código do Delphi. Para cada Form é criado um código, que é gravado em arquivos chamados Units. Nesses arquivos é definida a classe do Form e seus métodos de eventos. Para alternar entre o Form e sua Unit podemos clicar

em Toggle Form/Unit no menu View, ou no botão corresponde da Barra de Ferrramentas. Para cada Form aberto é criado um Form Designer e uma nova guia no Editor de Código.

- **Configuração do Ambiente** - grande parte das opções de configuração do ambiente podem ser acessadas através do item Environment Options do menu Tools. A maioria das opções desse diálogo são bastante claras e através delas podemos definir, desde as opções do Form Designer, até o Editor de Código e o caminho das Livrarias. No menu Tools, podemos escolher também Configure Tools, para permitir abrir aplicações externas a partir do ambiente do Delphi, como o Image Editor e o Database Desktop.
- **Project Manager -** para ajudar no gerenciamento de projetos, podemos usar o Project Manager pelo menu View. O Project Manager lista as Units, os Forms existentes nessas Units e o path, se a Unit não estiver na pasta do projeto. Através dos botões do Project Manager você pode adicionar, excluir e visualizar Units e Forms que compõem o projeto.
- **Project Options -** através do item Options, do menu Project, podemos escolher diversos aspectos de um projeto.
- **Forms -** nessa página, podemos definir o Form principal da aplicação e a os Forms que serão criados automaticamente. Se um Form não for criado automaticamente, você terá que instanciar esse Form explicitamente.
- **Application -** nessa página podemos definir o título, o arquivo help e o ícone da aplicação.
- **Compiler -** usamos essa página para definir as opções de compilação, para o projeto atual. Essas opções irão interferir diretamente no executável gerado.
- **Linker -** essa página é muito pouco usada, mas somente através dela podem modificar a memória exigida por uma aplicação.
- **Directories/Conditionals -** pode-se especificar pastas de saída para os arquivos gerados na compilação do projeto e opções de compilação condicional.
- **Version Information -** informações da versão do executável.
- **Packages -** Nesta página você pode especificar parte do código para ser incluído em *Packages*, fora do executável, permitindo compartilhamento de componentes entre várias aplicações Delphi.
- **Gerenciamento de Projetos -** segue uma descrição das mais importantes opções de menu para o gerenciamento de projetos, algumas dessas opções tem um botão correspondente na barra de ferramentas.

| **File** | |
|---|---|
| New | Abre um diálogo com novos itens que podem ser adicionados ao projeto |
| Open | Abrir projetos, pode abrir também Units, Forms e texto no editor de código |
| Save | Salva o arquivo aberto no editor de código |
| Save Project As | Salva o projeto com outro nome ou local |
| Use Unit | Faz com que a Unit atual possa usar outra Unit do projeto |
| Add to Project | Adiciona uma Unit em disco ao projeto |
| Remove from Project | Remove uma Unit do projeto |
| **View** | |
| Project Manager | Mostra o gerenciador de projeto |
| Project Source | Mostra o código do projeto |
| Object Inspector | Mostra o Object Inspector |
| Toggle Form/Unit | Alterna entre o Form e a Unit |
| Units | Mostra o código fonte de uma Unit ou do Projeto a partir de uma lista |
| Forms | Seleciona um Form a partir de uma lista |
| **Project** | |
| Compile | Compila o projeto |
| Options | Opções do projeto, como ícone do executável, nome da aplicação e opções de compilação |
| **Run** | |
| Run | Compila e executa o projeto |

- **Ajuda -** o sistema de ajuda do Delphi é a referência mais completa, seguida pelos manuais do usuário cedidos com o sistema. Se quiser ajuda sobre um componente, selecione-o e aperte F1, o mesmo pode ser feito com propriedades e eventos, no Object Inpector e comandos, no editor de código.

# 3 FUNDAMENTOS DE Object Pascal

## *3.1 Conceitos de Programação Orientada a Objetos*

Antes de partir para a linguagem propriamente dita, devemos considerar, de forma prática, alguns conceitos básicos de Programação Orientada a Objetos.
**Classe:** definição de tipo dos objetos, modelo de objeto.
**Objeto:** instância de classe, variável cujo tipo é uma classe.
**Atributos:** variáveis de instância que são os dados de um objeto.
**Métodos:** funções e procedimentos de um objeto.
**Propriedades:** *apelido* usado para evitar o acesso direto aos atributos de um objeto, onde podemos especificar métodos que serão usados para ler e atribuir seus valores a esses atributos.
**Mensagens:** chamada de métodos, leitura e atribuição de propriedades.
**Encapsulamento:** conjunto de técnicas usadas para limitar o acesso aos atributos e métodos internos de um objeto.
**Herança:** possibilidade de criar uma classe descendente de outra, aproveitando seus métodos, atributos e propriedades.
**Ancestral:** super classe ou classe de base, a partir da qual outras classes podem ser criadas.
**Descendente:** subclasse.
**Hierarquia de Classes:** conjunto de classes ancestrais e descendentes, geralmente representadas em uma árvore hierárquica.
**Polimorfismo:** capacidade de redefinir métodos e propriedades de uma classe em seus descendentes.

## *3.2 Variáveis*

No Delphi, toda variável tem que ser declarada antes de ser utilizada. As declarações podem ser feitas após a palavra reservada *var*, onde são indicados o nome e o tipo da variável. Os nomes de variáveis não podem ter acentos, espaços ou caracteres especiais como &, $ ou % e o primeiro caractere de um nome de variável tem que ser uma letra ou um sublinhado. O Delphi ignora o caso das letras.

### Variáveis Globais

As variáveis abaixo são globais, declaradas da *Interface* da Unit. Podem ser acessadas por qualquer Unit usuária.

```
var
  I: Integer;
  Usuario: string;
  A, B, Soma: Double;
  Ok: Boolean;
```

### Variáveis Locais

As variáveis abaixo são locais ao método, ou seja, elas só existem dentro do método, não podem ser acessadas de fora, mesmo que seja na mesma Unit. Na verdade essas variáveis são criadas quando o método é chamado e destruído quando ele é encerrado, seu valor não é persistente.

```
procedure TFrmExemplo.BtnTrocarClick(Sender: TObject);
var
  Aux: string;
begin
  Aux :=  EdtA.Text;
  EdtA.Text := EdtB.Text;
  EdtB.Text := Aux;
end;
```

## 3.3 Atributos

Os atributos são variáveis de instância. Para declarar um atributo em uma classe basta definir o identificador e o tipo do atributo na declaração da classe, feita na seção *type* da *Interface* da Unit, como abaixo.

```
type
  TFrmSomar = class(TForm)
  private
    { Private declarations }
    A, B: Double;
  public
    { Public declarations }
    Soma: Double;
  end;
```

## 3.4 Encapsulamento

Os principais níveis de visibilidade dos atributos e métodos de uma classe são mostrados abaixo.

| **Nível** | **Visibilidade** |
|---|---|
| Private | Os itens declarados nesse nível só podem ser acessados na mesma unit. |
| Public | Nesse nível, qualquer unit usuária poderá acessar o item. |
| Protected | Os itens só poderão ser acessados em outra unit se for em uma classe descendente |
| Published | É o nível default, igual ao Public, mas define propriedades e eventos usados em tempo de projeto. |

## 3.5 Classes

Classes são tipos de objetos, uma classe é declarada na cláusula type da seção interface e os métodos são definidos na seção implementation. Examine o código de um Form para identificar os elementos de sua classe.

```
interface
type
  TFrmSomar = class(TForm)
    EdtA: TEdit;
    EdtB: TEdit;
    BtnSoma: TButton;
    procedure BtnSomaClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

implementation

procedure TFrmSoma.BtnSomaClick(Sender: TObject);
begin
  ShowMessage(EdtA.Text + EditB.Text);
end;
```

## 3.6 Objetos

Um Objeto é tratado como uma variável cujo tipo é uma classe. A declaração de objetos é igual à declaração de uma variável simples, tendo no lugar do tipo a classe do objeto.

```
var
  FrmSomar: TFrmSomar;
```

## 3.7 Literais

Valores literais são valores usados em atribuições e expressões. Cada tipo tem uma sintaxe diferente.

| Tipo | Definição |
|---|---|
| Inteiro | Seqüência de dígitos decimais (0 a 9), sinalizados ou não |
| Inteiro Hexadecimal | Seqüência de dígitos hexadecimais (0 a F), precedidos por um sifrão ($) |
| Real | Igual ao tipo Inteiro, mas pode usar separador decimal e notação científica |
| Caractere | Letra entre apóstrofos ou o caracter # seguido de um número inteiro entre 0 e 255 (ASCII) |
| String | Seqüência de caracteres delimitados por apóstrofos |

## 3.8 Constantes

São declaradas na seção *const*, podem ser usadas como variáveis, mas não podem ser alteradas. Geralmente o nome das constantes é escrito em letras maiúsculas e na declaração dessas constantes não é indicado o tipo.

```
const
  G = 3.94851265E-19;
  NUM_CHARS = '0123456789';
  CR = #13;
  SPACE = ' ';
  MAX_VALUE = $FFFFFFFF;
```

### Constantes Tipadas

Na verdade, constantes tipadas são variáveis inicializadas com valor persistente, que podem ser alteradas normalmente, como qualquer variável. A única diferença de sintaxe entre constantes tipadas e simples é que o tipo da constante é indicado explicitamente na declaração. Se uma constante tipada for declarada localmente, ela não será destruída quando o método for encerrado. Para diferenciar das constantes normais, costuma-se declarar estas com letras de caso variável, como abaixo.

```
const
  Cont: Integer = 1;
  Peso: Double = 50.5;
  Empresa: string = 'SENAC';
```

## 3.9 Instruções

Os programas são compostos por instruções, que são linhas de código executável. Exemplos de instruções simples são atribuições, mensagens entre objetos, chamadas de procedimentos, funções e métodos, como mostradas abaixo. As instruções podem ser divididas em várias linhas, o que indica o fim de uma instrução é o ponto e vírgula no final. Quando uma instrução é quebrada, costuma-se dar dois espaços antes das próximas linhas, para melhorar a leitura do código.

```
Caption := 'Gabba Gabba Hey!';
Form2.ShowModal;
Application.MessageBox('Você executou uma operação ilegal, o programa será
finalizado.', 'Falha geral', MB_ICONERROR);
```

É possível se usar várias instruções agrupadas em uma instrução composta, como se fosse uma só instrução. Uma instrução composta delimitada pelas palavras reservadas *begin* e *end*. Toda instrução, simples ou composta, é terminada com um ponto-e-vírgula.

```
if CheckBox1.Checked then
begin
  ShowMessage('O CheckBox será desmarcado.');
  CheckBox1.Checked := False;
end;
```

### Estilo de Codificação

As instruções e todo o código de uma Unit devem ser distribuídos para facilitar o máximo a leitura. Para isso, podemos usar a indentação, geralmente de dois espaços para indicar os níveis de código. Procure criar um estilo próprio, que melhor se molde à sua realidade. Se for desenvolver em grupo, é melhor que todos usem o mesmo estilo para evitar confusões.

## *3.10 Comentários*

Existem três estilos de comentário no Delphi, como mostrado abaixo.

```
(* Comentário do Pascal Padrão *)
{ Comentário do Turbo Pascal }
// Comentário de linha do C++
```

Cuidado com as diretivas de compilação, pois elas são delimitadas por chaves e podem ser confundidas com comentários. A diretiva de compilação mostrada abaixo é incluída em todas as Units de Forms.

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

## *3.11 Tipos de Dados Padrão*

O Delphi trata vários tipos de dados padrão, segue uma descrição sucinta desses tipos.

### Tipos Inteiros

São tipos numéricos exatos, sem casas decimais. O tipo *Integer* é o tipo inteiro padrão.

| Tipo | Tamanho em Bytes | Valor Mínimo | Valor Máximo |
|---|---|---|---|
| ShortInt | 1 | -128 | 127 |
| SmallInt | 2 | -32768 | 32767 |
| Longint | 4 | -2147483648 | 2147483647 |
| Byte | 1 | 0 | 255 |
| Word | 2 | 0 | 65535 |
| Integer | 4 | -2147483648 | 2147483647 |
| Cardinal | 4 | 0 | 2147483647 |

### Tipos Reais

São tipos numéricos com casas decimais. O tipo *Double* é o tipo real padrão.

| Tipo | Tamanho em Bytes | Valor Mínimo | Valor Máximo | Dígitos Significativos |
|---|---|---|---|---|
| Real | 6 | $10^{-39}$ | $10^{38}$ | 11-12 |
| Single | 4 | $10^{-45}$ | $10^{38}$ | 7-8 |
| Double | 8 | $10^{-324}$ | $10^{308}$ | 15-16 |
| Extended | 10 | $10^{-4932}$ | $10^{4932}$ | 19-20 |
| Comp | 8 | $-10^{18}$ | $10^{18}$ | 19-20 |
| Currency | 8 | $-10^{12}$ | $10^{12}$ | 19-20 |

### Tipos Texto

Os tipos texto podem operar com caracteres simples ou grupos de caracteres. O tipo texto padrão é o tipo string.

| Tipo | Descrição |
|---|---|
| Char | Um único caractere ASCII |
| String | Texto alocado dinamicamente, pode ser limitado a 255 caracteres conforme configuração |
| PChar | String terminada em nulo (#0), usada geralmente nas funções da API do Windows |

O operador + pode ser usado para concatenar strings e você pode usar uma variável do tipo string como uma lista de caracteres.

```
ShowMessage('5ª letra do título da janela: ' + Caption[5]);
Label1.Text := '2ª letra do Edit: ' + Edit1.Text[2];
```

Existem várias funções de manipulação de strings, veja algumas das mais importantes mostradas abaixo.

| Função | Descrição |
|---|---|
| AnsiCompareText | Compara 2 strings sem sensitividade de maiúsculas/minúsculas |
| AnsiLowerCase | Converte todas as letras de uma string para minúsculas |
| AnsiUpperCase | Converte todas as letras de uma string para maiúsculas |
| Copy | Retorna parte de uma string |
| Delete | Apaga parte de uma string |
| Insert | Insere uma string em outra |
| Length | Número de caracteres de uma string |
| Pos | Posição de uma string em outra |
| Trim | Remove todos os espaços de uma string |
| TrimLeft | Remove os espaços à esquerda de uma string |
| TrimRight | Remove os espaços à direita de uma string |
| Format | Formata uma string com uma série de argumentos de vários tipos |

Por exemplo, para comparar o texto de dois Edits, poderíamos usar a função AnsiCompareText.

```
if AnsiCompareText(EdtA.Text, EdtB.Text) = 0 then
  ShowMessage('O texto dos dois Edits são iguais.');
```

A função Format é especialmente útil na formatação de strings, veja alguns exemplos.

```
ShowMessage(Format('O número %d é a parte inteira do número %f.', [10, 10.5]));
ShowMessage(Format('Este texto%sfoi formatado%susando o caractere #%d.', [#13, #13,
13]));
ShowMessage(Format('O preço do livro %s é %m.', ['Como Programar em Delphi',
50.7]));
```

Um detalhe que deve ser observado é que as propriedades dos objetos não podem ser usadas como variáveis em funções. Veja a declaração do procedimento Delete no help.

```
procedure Delete(var S: string; Index, Count:Integer);
```

Digamos que você deseje apagar as 5 primeiras letras de um Edit, como a string do Delete é variável, não poderia usar o código abaixo.

```
Delete(Edit1.Text, 1, 5);
```

Para você poder fazer a operação desejada, teria que usar uma variável como variável auxiliar.

```
var
  S: string;
begin
  S := Edit1.Text;
  Delete(S, 1, 5);
  Edit1.Text := S;
end;
```

## Tipos Ordinais

Tipos ordinais são tipos que tem uma seqüência incremental, ou seja, você sempre pode dizer qual o próximo valor ou qual o valor anterior a um determinado valor desses tipos. São tipos ordinais o Char,

os tipos inteiros, o Boolean e os tipos enumerados. Algumas rotinas para ordinais são mostradas abaixo.

| Função | Descrição |
|---|---|
| Dec | Decrementa variável ordinal |
| Inc | Incrementa variável ordinal |
| Odd | Testa se um ordinal é ímpar |
| Pred | Predecessor do ordinal |
| Succ | Sucessor do ordinal |
| Ord | Ordem de um valor na faixa de valores de um tipo ordinal |
| Low | Valor mais baixo na faixa de valores |
| High | Valor mais alto na faixa de valores |

Por exemplo, use o código abaixo no evento OnKeyPress de um Edit e veja o resultado.

```
Inc(Key);
```

## Boolean

Variáveis do tipo Boolean podem receber os valores lógicos True ou False, verdadeiro ou falso. Uma variável Boolean ocupa 1 byte de memória.

## TDateTime

O tipo TDateTime guarda data e hora em uma estrutura interna igual ao tipo Double, onde a parte inteira é o número de dias desde 31/12/1899 e a parte decimal guarda a hora, minuto, segundo e milissegundo. As datas podem ser somadas ou subtraídas normalmente.

Existem várias rotinas de manipulação de datas e horas, usadas com o tipo TDateTime, veja algumas a seguir.

| Rotina | Descrição |
|---|---|
| Date | Retorna a data do sistema |
| Now | Retorna a data e hora do sistema |
| Time | Retorna a hora do sistema |
| DayOfWeek | Retorna o dia da semana de uma data especificada |
| DecodeDate | Decodifica um valor TDateTime em Words de dia, mês e ano |
| DecodeTime | Decodifica um valor TDateTime em Words de hora, minuto, segundo e milissegundos |
| EncodeDate | Retorna um TDateTime a partir de Words de dia, mês e ano |
| EncodeTime | Retorna um TDateTime a partir de Words de hora, minuto, segundo e milissegundos |

No help de cada uma das funções acima você vai encontrar alguns exemplos, veja os colocados abaixo.

```
if DayOfWeek(Date) = 1 then
  ShowMessage('Hoje é Domingo, pé de cachimbo!')
else
  ShowMessage('Hoje não é Domingo, pé de cachimbo!');

var
  A, M, D: Word;
begin
  DecodeDate(Date, A, M, D);
  ShowMessage(Format('Dia %.2d do mês %.2d de %d.', [D, M, A]));
end;
```

### Variant

Tipo genérico, que pode atribuir e receber valores de qualquer outro tipo. Evite usar variáveis do tipo Variant, pois o uso dessas variáveis pode prejudicar o desempenho do programa, além de diminuir a legibilidade do código fonte e a integridade do executável, veja o trecho de código abaixo e note como esse tipo de variável tem um comportamento estranho.

```
var
  V1, V2, V3: Variant;
begin
  V1 := True;
  V2 := 1234.5678;
  V3 := Date;
  ShowMessage(V1 + V2 + V3);
end;
```

## *3.12 Conversões de Tipo*

Freqüentemente você vai precisar converter um tipo de dado em outro, como um número em uma string. Para essas conversões você pode usar duas técnicas, o TypeCasting e as rotinas de conversão de tipos.

**TypeCasting -** é uma conversão direta de tipo, usando o identificador do tipo destino como se fosse uma função. Como o Delphi não faz nenhuma verificação se a conversão é válida, você deve tomar um certo cuidado ao usar um TypeCast para não criar programas instáveis.

```
var
  I: Integer;
  C: Char;
  B: Boolean;
begin
  I := Integer('A');
  C := Char(48);
  B := Boolean(0);
  Application.MessageBox(PChar('Linguagem de Programação' + #13 + 'Delphi 3'),
'SENAC',
    MB_ICONEXCLAMATION);
end;
```

### Rotinas de Conversão

As principais rotinas de conversão estão listadas na tabela abaixo. Caso você tente usar uma dessas rotinas em uma conversão inválida, pode ser gerada uma exceção.

| Rotina | Descrição |
|---|---|
| Chr | Byte em Char |
| StrToInt | String em Integer |
| IntToStr | Integer em String |
| StrToIntDef | String em Integer, com um valor default caso haja erro |
| IntToHex | Número em String Hexadecimal |
| Round | Arredonda um número real em um Integer |
| Trunc | Trunca um número real em um Integer |
| StrToFloat | String em Real |
| FloatToStr | Real em string |
| FormatFloat | Número real em string usando uma string de formato |
| DateToStr | TDateTime em string de data, de acordo com as opções do Painel de Controle |
| StrToDate | String de data em TDateTime |
| TimeToStr | TDateTime em Strind de Hora |
| StrToTime | String de hora em TDateTime |
| DateTimeToStr | TDateTime em string de data e hora |
| StrToDateTime | String de data e hora em TDateTime |
| FormatDateTime | TDateTime em string usando uma string de formato |

| | |
|---|---|
| VarCast | Qualquer tipo em outro usando argumentos do tipo Variant |
| VarAsType | Variante em qualquer tipo |
| Val | String em número, real ou inteiro |
| Str | Número, real ou inteiro, em String |

Veja alguns exemplos de como usar essas rotinas. Conversão de dados é uma operação muito comum na programação em Object Pascal, seeria interessante dar uma olhada no help de cada uma das funções acima.

```
var
  I: Integer;
  D: Double;
  S1, S2: string;
begin
  D := 10.5;
  I := Trunc(D);
  S1 := FloatToStr(D);
  S2 := IntToStr(I);
  ShowMessage(S1 + #13 + S2);
end;

var
  A, B, Soma: Bouble;
begin
  A := StrToFloat(EdtA.Text);
  B := StrToFloat(EdtB.Text);
  Soma := A + B;
  ShowMessage(Format('%f + %f = %f', [A, B, Soma]);
end;
```

## *3.13 Expressões*

Uma expressão é qualquer combinação de operadores, variáveis, constantes, valores literais e chamadas de funções que resultem em um valor de determinado tipo. Uma expressão é usada sempre que precisamos de um valor que possa ser obtido por uma expressão. Alguns exemplos são apresentados a seguir.

```
A + 12 * C
Date - 4
StrToInt(Edit1.Text + Edit2.Text)
StrToDate(Edit2.Text) - StrToDate(Edit1.Text)
12 * A / 100
A < B
```

## *3.14 Operadores*

Os operadores são usados em expressões e a ordem em que as expressões são executadas depende da precedência desses operadores. Veja abaixo a lista de operadores em ordem descendente de precedência.

| **Operadores Unários** | |
|---|---|
| @ | Endereço |
| not | *Não* booleano ou bit voltado para *não* |
| **Operadores Multiplicativos e de direção de Bit** | |
| * | Multiplicação ou interseção de conjuntos |
| / | Divisão de Real |
| div | Divisão de Inteiro |
| mod | Resto de divisão de Inteiros |
| as | TypeCast seguro quanto ao tipo (RTTI) |

| | |
|---|---|
| and | *E* booleano ou bit voltado para *e* |
| shl | Deslocamento de bits à esquerda |
| shr | Deslocamento de bits à direita |
| **Operadores Aditivos** | |
| + | Adição ou união de conjuntos |
| - | Subtração ou diferença de conjuntos |
| or | *Ou* booleano ou bit voltado para *ou* |
| xor | *Ou* exclusivo booleano ou bit voltado para *ou* exclusivo |
| **Operadores Relacionais** | |
| = | Igual |
| <> | Diferente |
| < | Menor |
| > | Maoir |
| <= | Menor ou igual |
| >= | Maior ou igual |
| in | Pertinência a conjuntos |
| is | Compatibilidade de tipos (RTTI) |

Para forçar uma expressão de menor precedência a ser executada antes, você pode usar os parênteses, como mostrado abaixo.

```
(5 - 2) * 3;
(A > B) and (A < C)
```

Para fazer potenciação, use a função *Power*, abaixo temos que A é igual a A elevado a 4.

```
A := Power(A, 4);
```

## 3.15 Estruturas de Decisão

**If** - uma estrutura de decisão usada para realizar instruções em determinadas condições. O if é considerado uma só instrução, por isso, só encontramos o ponto-e-vírgula no final. O else é opcional.

```
if Opn.Execute then
  Img.Picture.LoadFromFile(Opn.FileName);

if Nota < 5 then
  ShowMessage('Reprovado')
else
  ShowMessage('Aprovado');
```

**Case** - permite que o fluxo da execução seja desviado em função de várias condições de acordo com o valor do argumento, que tem que ser ordinal, caso o valor do argumento não corresponda a nenhum dos valores listados, podemos incluir um else.

```
case Ch of
  ' ': ShowMessage('Espaço');
  '0'..'9': ShowMessage('Dígito');
  '+', '-', '*', '/': ShowMessage('Operador');
else
  ShowMessage('Caractere especial');
end;

case CbbBorda.ItemIndex of
  0: BorderStyle := bsDialog;
  1: BorderStyle := bsSingle;
  2: BorderStyle := bsSizeable;
end;
```

## 3.16 Estruturas de Repetição

**While** - executa uma instrução até que uma condição deixe de ser verdadeira.

```
I := 10;
while I >= 0 do
begin
  ShowMessage(IntToStr(I));
  Dec(I);
end;
```

**For** - executa uma instrução um número determinado de vezes, incrementando uma variável de controle automaticamente a cada iteração. Caso seja preciso que a contagem seja decremental, pode-se usar *downto* em vez de *to*.

```
for I := 1 to ComponentCount do
  ShowMessage('O ' + IntToStr(I) + 'º Componente é ' + Components[I -
1].Name);

for I := Length(Edit1.Text) downto 1 do
  ShowMessage(Edit1.Text[I]);
```

**Repeat** - executa instruções até que uma condição seja verdadeira.

```
I := 1;
repeat
  S := InputBox('Acesso', 'Digite a senha', '');
  Inc(I);
  if I > 3 then
    Halt;
until S = 'fluminense';
```

**Quebras de Laço** - em qualquer um dos laços mostrados podemos usar o procedimento Break para cancelar a repetição e sair do laço, podemos também forçar a próxima iteração com o procedimento Continue.

```
I := 1;
while true do
begin
  Inc(I);
  if I < 10000000 then
    Continue;
  ShowMessage('Chegamos a dez milhões');
  Break;
end;
```

## *3.17 Tipos Definidos pelo Usuário*

O usuário também pode declarar tipos não definidos pelo Delphi. Essas declarações são feitas na seção *type*, da interface ou implementation, sendo que na implementation esses tipos não poderão ser usados em outras Units. Mas, de fato, dificilmente você terá que definir tipos, a não ser classes, pois os tipos padrão do Delphi são o bastante para a maioria das aplicações.

**Strings Limitadas -** caso se deseje limitar o número de caracteres que uma string pode receber, podemos criar um tipo de string limitada.

```
TNome = string[40];
TEstado = string[2];
```

**Tipo Sub-Faixa -** É um subconjunto de um tipo ordinal e possui as mesmas propriedades do tipo original**.**

```
TMaiusculas = 'A'..'Z';
TMes = 1..12;
```

**Enumerações -** define uma seqüência de identificadores como valores válidos para o tipo. A cada elemento da lista de identificadores é associado internamente um número inteiro, iniciando pelo número 0, por isso são chamados de tipos enumerados.

```
TBorderIcon = (biSystemMenu, biMinimize, biMaximize, biHelp);
TDiaSemana = (Seg, Ter, Qua, Qui, Sex, Sab, Dom);
```

**Ponteiros** - armazenam endereços de memória, todas as classes em Object Pascal são implementadas como ponteiros:

```
TIntPtr: ^Integer;
Quanto desejo o ponteiro de um outro objeto:
TIntPtr := @Tvarnum;
Quanto desejo manipular o objeto e não o ponteiro:
showmessage ( IntToStr ( TIntPtr^ ) );
```

**Records -** o tipo record é uma forma de criar uma única estrutura com valores de diferentes tipos de dados. Cada um dos dados de um record é chamado de campo.

```
TData = record
  Ano: Integer;
  Mes: TMes;
  Dia: Byte;
end;
var
  Festa: TData;
begin
  Festa.Ano := 1997;
  Festa.Mes := Mai;
  Festa.Dia := 8;
end;
```

**Arrays -** Fornecem uma forma de criar variáveis que contenham múltiplos valores, como em uma lista ou tabela, cujos elementos são do mesmo tipo. A seguir, alguns exemplos de arrays de dimensões variadas.

```
TTempDia = array [1..24] of Integer;
TTempMes = array [1..31, 1..24] of Integer;
TTempAno = array [1..12, 1..31, 1..24] of Integer;
var
  TD: TTempDia;
  I: Integer;
begin
  for I := 1 to 24 do
    TD[I] := StrToIntDef(InputBox('Temperaturas', 'Digite a temperatura na
hora '
      + IntToStr(I), ''), 30);
end;
```

Um array pode ser definido como constante tipada, onde todos os seus elementos devem ser inicializados.

```
FAT: array[1..7] of Integer = (1, 2, 6, 24, 120, 720, 5040);
```

O tipo dos elementos de um array pode ser qualquer um, você pode ter uma array de objetos, de conjuntos, de qualquer tipo que quiser, até mesmo um array de arrays.

```
TTempMes = array [1..31] of TTempDia;
TBtnList = array [1..10] of TButton;
```

**Sets -** São conjuntos de dados de um mesmo tipo, sem ordem, como os conjuntos matemáticos. Conjuntos podem conter apenas valores ordinais, o menor que um elemento pode assumir é zero e o maior, 255.

```
TBorderIcons = set of BorderIcon;
```

```
BorderIcons := [biSystemMenu, biMinimize];

if MesAtual in [Jul, Jan, Fev] then
  ShowMessage('Férias');
```

Os conjuntos podem ser definidos como constantes ou constantes tipadas, como abaixo.

```
DIG_HEXA = ['0'..'9', 'A'..'Z', 'a'..'z'];
DIG_HEXA: set of Char = ['0'..'9', 'A'..'Z', 'a'..'z'];
```

## *3.18 Procedimentos, Funções e Métodos*

As ações de um objeto devem ser definidas como métodos. Quando a ação não pertence a um objeto, como uma transformação de tipo, essa ação deve ser implementada em forma de procedimentos e/ou funções.

### Procedimentos

São sub-rotinas, que realizam uma tarefa e não retornam um valor. A declaração de um procedimento é feita na seção interface e a definição, na seção implementation. Ao chamar o identificador do procedimento, com os parâmetros necessários, esse procedimento será executado. Veja abaixo o exemplo de uma unit com a implementação um procedimento.

```
unit Tools;
interface
  procedure ErrorMsg(const Msg: string);

implementation
uses Forms, Windows;

procedure ErrorMsg(const Msg: string);
begin
  Application.MessageBox(PChar(Msg), 'Operação inválida', MB_ICONERROR);
end;
end.
```

### Funções

São muito semelhantes com procedimentos; a única diferença é que as funções retornam um valor. O tipo do valor de retorno deve ser informado no cabeçalho da função. Na implementação da função deve-se atribuir o valor de retorno à palavra reservada *Result* ou ao identificador da função. Pode-se então usar a função em expressões, atribuições, como parâmetros para outras funções, em qualquer lugar onde o seu valor possa ser usado.

```
function Average(A, B: Double): Double;
begin
  Result := (A + B) / 2;
end;
```

### Métodos

São funções ou procedimentos que pertencem a alguma classe, passando a fazer parte de qualquer objeto dessa classe. Na implementação de um método precisamos indicar qual a classe à qual ele pertence. Para chamar um método em algum lugar não pertencente à sua classe, como procedimentos, funções ou métodos de outras classes, deve ser indicado o objeto que deve executar o método. Os métodos usam os mesmos níveis de encapsulamento dos atributos.

```
type
  TFrmMsg = class(TForm)
    LblMsg: TLabel;
    BtnOk: TButton;
    BtnCancelar: TButton;
    ImgMsg: TImage;
  public
```

```
    procedure ShowMsg(const Msg: string);
  end;

procedure TFormMsg.ShowMsg(const Msg: string);
begin
  LblMsg.Caption := Msg;
  ShowModal;
end;
```

### Parâmetros

Existem três tipos de passagem de parâmetros, que devem ser indicados na declaração da função ou procedimento. Parâmetros de tipos diferentes de vem ser separados por ponto e vírgula.

```
function MultiStr(const S: string;  N: Double; var Erro: Integer): string;
```

- Quando não é indicado o tipo de passagem, é passado o valor do parâmetro, como constante.
- Ao usar a palavra-chave *var* não é enviado o valor do parâmetro, mas a referência a ele, tornando possível mudar o valor do parâmetro no código do procedimento.
- Como alternativa você pode passar um parâmetro por referência constante, para isso use a palavra const antes da declaração do parâmetro.

## *3.19 With*

Usado para facilitar o acesso às propriedades e métodos de um objeto.

```
with Edt do
begin
  CharCase := ecUpperCase;
  MaxLenght := 10;
  PasswordChar := '*';
  Text := 'Brasil';
end;
```

## *3.20 Self*

Self é usado quando se quer referenciar a instância atual da classe. Se você precisar referenciar a instância atual de uma classe, é preferível usar Self em vez de usar o identificador de um Objeto, isso faz com que o código continue funcionando para as demais instâncias da classe e em seus descendentes.

## *3.21 Criando e Destruindo Objetos*

Antes de tudo, você deve declarar o objeto, se quiser referenciá-lo. Para criá-lo, use o método Create, que é um método de classe. Para você usar um método de classe, referencie a classe, não o Objeto, como mostrado a seguir.

```
var
  Btn: TBitBtn;
begin
  Btn := TBitBtn.Create(Self);
  With Btn do
  begin
    Parent := Self;
    Kind := bkClose;
    Caption := '&Sair';
    Left := Self.ClientWidth - Width - 8;
    Top := Self.ClientHeight - Height - 8;
  end;
end;
```

Porém, se você não precisar referenciar o Objeto, poderia criar uma instância sem referência.

```
with TBitBtn.Create(Self) do
begin
  Parent := Self;
  Kind := bkClose;
  Caption := '&Sair';
  Left := Self.ClientWidth - Width - 8;
  Top := Self.ClientHeight - Height - 8;
end;
```

Para destruir um objeto, use o método Free. Para Forms, é recomendado usar o Release, para que todos os eventos sejam chamados. O parâmetro do método Create é usado apenas em Componentes, para identificar o componente dono. Ao criar Forms, poderíamos usar o Objeto Application.

```
FrmSobre := TFrmSobre.Create(Application);
FrmSobre.ShowModal;
FrmSobre.Release;
```

Para criar objetos não componentes, você não precisa de nenhum parâmetro no método Create.

```
var
  Lst: TStringList;
begin
  Lst := TStringList.Create;
  Lst.Add('Alô, Teresinha!');
  Lst.Add('Uhh uhh...');
  Lst.SaveToFile('Teresinha.txt');
  Lst.Free;
end;
```

## 3.22 RTTI

Run Time Type Information é a informação de tipo dos objetos em tempo de execução. O operador *is* é usado para fazer comparações e o operador *as* é usado para fazer um TypeCast seguro com objetos.

```
for I := 0 to ComponentCount - 1 do
  if Components[I] is TEdit then
    TEdit(Components[I]).Clear;

(Sender as TEdit).Color := clYellow;
```

## 3.23 Exceções

O tratamento de exceção é um mecanismo capaz de dar robustez a uma aplicação, permitindo que os erros sejam manipulados de uma maneira consistente e fazendo com que a aplicação possa se recuperar de erros, se possível, ou finalizar a execução quando necessário, sem perda de dados ou recursos.

Para que uma aplicação seja segura, seu código necessita reconhecer uma exceção quando esta ocorrer e responder adequadamente a essa exceção. Se não houver tratamento para uma exceção, será exibida uma mensagem padrão descrevendo o erro e todos os processamentos pendentes não serão executados. Uma exceção deve ser respondida sempre que houver perigo de perda de dados ou de recursos do sistema.

Exceções são classes definidas pelo Delphi para o tratamento de erros. Quando uma exceção é criada, todos os procedimentos pendentes são cancelados e, geralmente é mostrada uma mensagem de erro para o usuário. As mensagens padrão nem sempre são claras, por isso é indicado criar seus próprios blocos protegidos.

## Blocos Protegidos

Um bloco protegido é um grupo de comandos com uma seção de tratamento de exceções.

```
try
  A := StrToFloat(EdtA.Text);
  B := StrToFloat(EdtB.Text);
  ShowMessage(Format('%f / %f = %f', [A, B, A + B]));
except
  ShowMessage('Números inválidos.');
end;
```

Algumas vezes você pode precisar especificar quais exceções quer tratar, como mostrado abaixo.

```
try
  Soma := StrToFloat(EdtSoma.Text);
  NumAlunos := StrToInt(EdtNum.Text);
  ShowMessage(Format('Média igual a %f.', [Soma / NumAlunos]));
except
  on EConvertError do
    ShowMessage('Valor inválido para soma ou número de alunos.');
  on EZeroDivide do
    ShowMessage('O número de alunos tem que ser maior que zero.');
  else
    ShowMessage('Erro na operação, verifique os valores digitados.');
end;
```

## Principais Exceções

O Delphi define muitas exceções, para cada erro existe uma exceção correspondente.

| Classe | Descrição |
|---|---|
| Exception | Exceção genérica, usada apenas como ancestral de todas as outras exceções. |
| EAbort | Exceção silenciosa pode ser gerada pelo procedimento Abort e não mostra nenhuma mensagem. |
| EAccessViolation | Acesso inválido à memória, geralmente ocorre com objetos não inicializados. |
| EConvertError | Erro de conversão de tipos. |
| EDivByZero | Divisão de inteiro por zero. |
| EInOutError | Erro de Entrada ou de Saída reportado pelo sistema operacional. |
| EIntOverFlow | Resultado de um cálculo inteiro excedeu o limite. |
| EInvalidCast | TypeCast inválido com o operador as. |
| EInvalidOp | Operação inválida com número de ponto flutuante. |
| EOutOfMemory | Memória insuficiente. |
| EOverflow | O resultado de um cálculo, com número real, excedeu o limite. |
| ERangeError | Valor excede o limite do tipo inteiro ao qual foi atribuída. |
| EUnderflow | O resultado de um cálculo, com número real, é menor que a faixa válida. |
| EVariantError | Erro em operação com variant. |
| EZeroDivide | Divisão de real por zero. |
| EDatabaseError | Erro genérico de banco de dados, geralmente não é usado diretamente. |
| EDBEngineError | Erro da BDE, descende de EDatabaseError e traz dados que podem identificar o erro. |

## Blocos de Finalização

Blocos de finalização são executados sempre, haja ou não uma exceção. Geralmente os blocos de finalização são usados para liberar recursos.

```
FrmSobre := TFrmSobre.Create(Application);
try
  FrmSobre.Img.LoadFromFile('Delphi.bmp');
  FrmSobre.ShowModal;
finally
  FrmSobre.Release;
end;
```

Você pode usar blocos de proteção e finalização aninhados

```
FrmOptions := TFrmOptions.Create(Application);
try
  FrmOptions.ShowModal;
  try
    Tbl.Edit;
    TblValor.AsString := EdtValor.Text;
  except
    on EDBEngineError do
      ShowMessage('Alteração não permitida.');
    on EConvertError do
      ShowMessage('Valor inválido.');
  end;
finally
  FrmOptions.Release;
end;
```

## Geração de Exceções

Você pode provocar uma exceção usando a cláusula *raise*.

```
raise EDatabaseError.Create('Erro ao alterar registro.');
```

Também é possível criar seus próprios tipos de exceções.

```
type
  EInvalidUser = class (Exception);
raise EInvalidUser.Create('Você não tem acesso a essa operação.');
```

Se você quiser que uma exceção continue ativa, mesmo depois de tratada, use a cláusula *raise* dentro do bloco de tratamento da exceção. Geralmente isso é feito com exceções aninhadas.

```
try
  Tbl.Edit;
  TblContador.Value := TblContador.Value + 1;
  Tbl.Post;
except
  ShowMessage('Erro ao alterar contador.');
  raise;
end;
```

## Erros de Bancos de Dados

A exceção EDBEngineError permite a identificação de erros de bancos de dados gerados pela BDE (*Borland Database Engine*).

```
try
  TblCli.Post;
except
  on E: EDBEngineError do
    if E.Errors[0].ErrorCode = DBIERR_KEYVIOL then
      ShowMessage('Cliente já cadastrado.');
end;
```

Note que a variável E, que vai identificar o erro, só precisa ser declarada no bloco de tratamento da exceção. No help você pode consultar outras propriedades de EDBEngineError que podem ser importantes.

Você também pode usar os eventos de erro do componente Table, sem precisar de blocos de tratamento.

```
procedure TFrmCadCli.TblCliPostError(DataSet: TDataSet; E: EDatabaseError;
  var Action: TDataAction);
begin
  if(E is EDBEngineError) then
    with EDBEngineError(E) do
```

```
      case Errors[0].ErrorCode of
        DBIERR_KEYVIOL: ShowMessage('Cliente já cadastrado.');
        DBIERR_REQDERR: ShowMessage('Campo obrigatório não preenchido.');
      end
  else
    ShowMessage('Erro no banco de dados:' + #13#13 + E.Message);
  Action := daAbort;
end;
```

Alguns códigos de erro da BDE estão listados abaixo. Todas as constantes e funções relacionadas à API da BDE no Delphi 3 estão na Unit BDE, que deve ser adicionada à cláusula uses. No BDE API Help você pode encontrar referência sobre as funções nativas da BDE, como também alguns exemplos em Delphi.

| Constante | Descrição |
|---|---|
| DBIERR_KEYVIOL | Violação de chave primária |
| DBIERR_MAXVALERR | Valor máximo excedido |
| DBIERR_FORIEGNKEYERR | Erro de chave externa, como em integridade referencial |
| DBIERR_LOCKED | Registro travado |
| DBIERR_FILELOCKED | Arquivo travado |
| DBIERR_NETMULTIPLE | Mais de um diretório usado como NetFileDir |
| DBIERR_MINVALERR | Campo com valor mais baixo que valor mínimo |
| DBIERR_REQDERR | Campo obrigatório faltando |
| DBIERR_LOOKUPTABLEERR | Erro em tabela Lookup |

Se você quiser mais informações a respeito do erro pode usar o procedimento DBIGetErrorContext, como na função mostrada abaixo que retorna determinadas informações sobre o erro.

```
function GetErrorInfo(Context: SmallInt): string;
begin
  SetLength(Result, DBIMAXMSGLEN + 1);
  try
    DbiGetErrorContext(Context, PChar(Result));
    SetLength(Result, StrLen(PChar(Result)));
  except
    Result := '';
  end;
end;
```

No evento OnEditError, usado no exemplo abaixo, se ocorrer um erro ao tentar alterar um registro, podemos identificar o usuário da rede que está alterando esse registro usando a função criada anteriormente.

```
if Pos('locked', E.Message) > 0 then
  ShowMessage('Usuário ''' + GetErrorInfo(ecUSERNAME) + ''' está alterando o
registro.');
```

Note que foi usada uma outra técnica de identificação do erro, usando a própria mensagem de erro e não o código, como mostrado anteriormente. Você pode usar a função criada acima mandando como parâmetro os valores mostrados abaixo, que podem ser encontrados no help da BDE.

| Constante | Descrição |
|---|---|
| ecTABLENAME | Nome da Tabela |
| EcFIELDNAME | Nome do campo |
| EcUSERNAME | Nome do usuário, muito usado para identificar qual usuário travou o registro |
| EcFILENAME | Nome do arquivo |
| EcINDEXNAME | Nome do índice |
| EcDIRNAME | Pasta |
| EcKEYNAME | Chave primária |
| EcALIAS | Alias |
| EcDRIVENAME | Drive |

| | |
|---|---|
| EcNATIVECODE | Código de erro nativo |
| EcNATIVEMSG | Mensagem de erro nativa |
| EcLINENUMBER | Número da linha, usado em instruções SQL |

Para desenvolver um sistema genérico de tratamento de erros, considere a opção de criar esse tratamento em um DataModule genérico para ser usado como ancestral por todos os DataModules do sistema, utilizando a herança visual.

Se o único problema for traduzir as mensagens, localize os arquivos CONSTS.INT e DBCONSTS.INT e crie uma nova Unit de definição de strings com uma estrutura semelhante a mostrada abaixo e juntando todas as definições das constantes das duas Units devidamente traduzidas. Depois, basta usar essa Unit em seus projetos que as novas mensagens irão sobrepor às anteriores.

```
unit NewConsts;

interface

resourcestring
  SAssignError = 'Não é possível atribuir %s a %s';
  SFCreateError = 'Não é possível criar arquivo %s';
  SFOpenError = 'Não é possível abrir arquivo %s';
  SInvalidFieldSize = 'Tamanho de campo inválido';
  SInvalidFieldRegistration = 'Registro de campo inválido';
  SUnknownFieldType = 'Campo ''%s'' tem um tipo desconhecido';

implementation

end.
```

Uma outra opção seria criar um método para o evento OnException do objeto Application, esse método seria chamado sempre que houvesse uma exceção em qualquer parte do sistema.

# 4 BIBLIOTECA DE CLASSES

## *4.1 Nomenclatura*

Para nomear os componentes podemos usar uma convenção muito usada, onde as primeiras letras, minúsculas, identificam o tipo do componente e o restante identifica a função deste, assim, btnSair, seria o botão de sair.

Se a função do componente for um nome composto esse nome deve ser escrito com os primeiros nomes abreviados e com letras de caso variável, como em btnRelVendas, que seria o botão do relatório de vendas ou btnRelVenProduto, que seria o botão do relatório de vendas por produto.

## *4.2 Propriedades*

As propriedades são características dos componentes, como foi mostrado anteriormente. Para alterar propriedades em código use a sintaxe de ponto, como mostrado a seguir.

### Tipos de Propriedade

- Tipo String

```
Button1.Caption := 'Fechar';
Label1.Caption := Edit1.Text + '/' + Edit2.Text;
```

- Tipo Numérico

```
Button2.Height := Button2.Height * 2;
Width := Button1.Width + Button2.Width + 12;
```

- Tipo Enumerado

```
BorderStyle := bsDialog;
```

```
Panel1.Color := clWindow;
```

- Propriedades Aninhadas de Classe

```
Memo1.Lines.Text := 'E agora, José?';
Label1.Font.Color := clBlue;
```

- Propriedades Aninhadas de Conjunto

```
BorderIcons := [biSystemMenu, biMaximize];
Label1.Font.Style := [fsBold, fsItalic];
```

## Propriedades Comuns

| Propriedade | Descrição |
|---|---|
| Align | Determina o alinhamento do componente |
| Canvas | Superfície de desenho, do tipo TCanvas, onde pode se desenhar a imagem do componente |
| Caption | Legenda do componente (& indica tecla de atalho para alguns componentes) |
| Color | Cor do componente |
| ComponentCount | O número de componentes possuídos |
| Components | Matriz de componentes possuídos |
| Ctl3D | Define a aparência 3D do componente |
| Enabled | Define se o componente está ativo, se pode ser usado |
| Font | Fonte utilizada no componente |
| Height | Altura |
| HelpContext | Número utilizado para chamar o Help on-line |
| Hint | String utilizada em dicas instantâneas |
| Left | Posição esquerda |
| Name | Nome do componente |
| PopupMenu | Menu de contexto do componente |
| ShowHint | Define se o Hint será mostrado |
| TabOrder | A ordem de tabulação do componente, usada quando o usuário tecla TAB |
| TabStop | Indica se o componente será selecionado quando o usuário teclar TAB |
| Tag | Propriedade não utilizada pelo Delphi, que pode ser usada como propriedade personalizada. |
| Top | Posição superior |
| Visible | Define se o componente está visível |
| Width | Largura |

## *4.3 Eventos*

Os Eventos acontecem em resposta a uma ação do usuário ou do próprio sistema, ao programar um método de evento, devemos levar em consideração que este só será executados quando o evento acontecer. Uma das tarefas mais importantes na programação baseada em eventos é determinar quais eventos serão usados e qual a ordem desses eventos, por exemplo, quando o usuário clicar em um botão, qual evento acontecerá primeiro, OnEnter, OnMouseDown ou OnClick?

Os eventos podem ser compartilhados entre componentes, dessa Forma, você pode ter um botão na barra de ferramentas que faz a mesma coisa que uma opção de menu. Para isso, basta escolher o evento na lista em vez de clicar duas vezes no Object Inspector.

Podemos também mudar os métodos de evento em código, pois os eventos também são propriedades e podem ser usados como tal. Você pode atribuir um evento de outro componente ou diretamente o nome do método, como mostrado abaixo.

```
Button1.OnClick := Edit1.OnExit;
Button2.OnClick := Edit2Click;
```

## Eventos Comuns

| Evento | Descrição |
|---|---|
| OnChange | O conteúdo do componente é alterado |
| OnClick | O componente é acionado |
| OnDblClick | Duplo-clique no componente |

| | |
|---|---|
| OnEnter | O componente recebe o foco |
| OnExit | O componente perde o foco |
| OnKeyDown | Tecla pressionada |
| OnKeyPress | Uma tecla é pressionada e solta |
| OnKeyUp | Tecla é solta |

## *4.4 Métodos*

Os métodos realizam ações definidas pelo componente, veja os exemplos abaixo e atente para os parâmetros passados. Note que podemos chamar os métodos de evento como qualquer outro método e que os métodos de evento pertencem ao Form, não aos componentes.

```
Edit1.Clear;
Form2.Show;
Close;
ScaleBy(110, 100);
Button1.ScrollBy(10, 10);
Button1.OnClick(Sender);
Button1Click(Self);
Form2.Button1Click(Sender);
```

### Métodos Comuns

| **Método** | **Descrição** |
|---|---|
| Create | Cria um novo Objeto de uma Classe |
| Free | Destrói um Objeto e libera a memória ocupada por ele |
| Show | Torna o componente visível |
| Hide | Torna o componente invisível |
| SetFocus | Coloca o foco no componente |
| Focused | Determina se o componente tem o foco |
| BringToFront | Coloca o componente na frente dos outros |
| SendToBack | Coloca o componente atrás dos outros |
| ScrollBy | Move o componente |
| ScaleBy | Gradua o componente em determina escala |
| SetBounds | Muda a posição e o tamanho do componente |

## *4.5 Janelas*

Todo aplicativo Windows é composto por janelas, que são o elemento básico no desenvolvimento Delphi, sobre o qual um aplicativo é construído. O tipo TForm é usado no Delphi como classe base para todas as janelas, veja abaixo algumas propriedades, eventos e métodos dessa classe.

| **Propriedade** | **Descrição** |
|---|---|
| Active | Indica se o Form está ativo |
| ActiveControl | Determina o controle que receberá o foco por default |
| AutoScroll | Adiciona barras de rolagem automaticamente, quando necessário |
| BorderIcons | Define quais ícones de controle serão visíveis, quais botões vão aparecer na barra de título |
| BorderStyle | Estilo da borda do Form |
| FormStyle | Tipo de Form, normal, MDI pai, MDI filho ou sempre visível |
| Icon | Ícone do Form |
| Menu | Indica qual o menu do Form |
| Position | Permite controlar a posição e tamanho do Form na exibição |
| WindowMenu | Automatiza o item de menu Window (MDI) |
| WindowState | Estado do Form, maximizada, minimizada ou normal |
| **Evento** | **Descrição** |
| OnCreate | Quando o Form é instanciado |
| OnDestroy | Quando o Form é liberado da memória |
| OnShow | Exatamente antes de mostrar o Form |
| OnCloseQuery | É chamada para validar se o Form pode ser fechado |

| | |
|---|---|
| OnClose | Quando o Form é fechado |
| OnActivate | Quando o Form recebe o foco |
| OnDeactivate | Quando o Form perde o foco |
| OnResize | Quando o Form muda de tamanho |
| **Método** | **Descrição** |
| Cascade | Organiza as Forms filhos em cascata (MDI) |
| Tile | Organiza as Forms filhos lado a lado (MDI) |
| ArrangeIcons | Organiza os ícones dos Forms Filhos minimizados (MDI) |
| ShowModal | Ativa o Form modal, que o usuário tem que fechar para poder continuar a usar a aplicação |
| Show | Mostra o Form |
| Close | Fecha o Form |
| Previous | Ativa o Form anterior (MDI) |
| Next | Ativa a próximo Form (MDI) |

## *4.6 Componentes Padrão*

**Tbutton -** Componente botão padrão do Windows, utilizado para executar ações.

| **Propriedade** | **Descrição** |
|---|---|
| Cancel | Dispara o evento OnClick do botão quando a tecla ESC é pressionada em qualquer controle |
| Default | Dispara o evento OnClick do botão quando a tecla ENTER é pressionada em qualquer controle |
| ModalResult | Associa o botão a opção de fechamento de um Form modal |
| **Método** | **Descrição** |
| Click | Ativa o evento OnClick do botão |

**TbitBtn** - Botão especializado, com Bitmap.

| **Propriedade** | **Descrição** |
|---|---|
| Glyph | Bitmap exibido pelo botão |
| LayOut | Posição do Bitmap no Botão |
| Margin | Indica o espaço entre a borda do botão e o Bitmap |
| Spacing | Indica o espaço entre o Bitmap e o texto do botão |
| Kind | Seleciona um tipo padrão para o botão, mudando várias propriedades, como Glyph e ModalResult |

**TspeedButton** - Botão com Bitmap, normalmente utilizado em barras de ferramentas.

| **Propriedade** | **Descrição** |
|---|---|
| Down | Estado do botão (Pressionado ou não) |
| GroupIndex | Indica quais botões pertencerão ao mesmo grupo |
| AllowAllUp | Permite que todos os botões de um grupo possam ficar não pressionados |
| Flat | Define se a borda do botão deve aparecer apenas quando ele for apontado |

**Tlabel** - Utilizado para exibir rótulos

| **Propriedade** | **Descrição** |
|---|---|
| Alignment | Alinhamento do texto no componente |
| AutoSize | Define se o tamanho do componente será automaticamente ajustado ao tamanho do Caption |
| WordWrap | Retorno automático de linha |
| Transparent | Define se o componente será transparente |
| FocusControl | Componente que receberá o foco quando a tecla de atalho do Caption (&) for pressionada |
| ShowAccelChar | Indica se o caractere & será usado para definir tecla de atalho |

**Tedit** - Utilizado para entrada de texto em uma única linha.

| **Propriedade** | **Descrição** |
|---|---|
| Text | Texto do componente |
| AutoSelect | Indica se o texto será ou não selecionado quando o componente receber o foco |
| MaxLength | Número máximo de caracteres permitidos |
| CharCase | Define se as letras aparecerão em maiúsculo, minúsculo ou normal |
| PasswordChar | Caractere utilizado para esconder o texto digitado (Senhas) |
| ReadOnly | Define se será permitido alterar o texto |
| **Método** | **Descrição** |
| Clear | Limpa o conteúdo do componente |
| ClearSelection | Limpa o texto selecionado no componente |

**TmaskEdit** - Permite entrada de dados texto em uma linha, utilizando uma máscara de edição. Possui todas as propriedades do componente TEdit.

| **Propriedade** | **Descrição** |
|---|---|
| EditMask | Máscara de edição |

**Máscaras** - uma máscara é composta por três partes, a primeira parte é a máscara propriamente dita, a segunda parte indica se os caracteres literais serão salvos e a terceira parte indica qual o caractere utilizado para representar os espaços a serem digitados no texto.

Estes são os caracteres especiais que podem compor a máscara de edição:

| **Caractere** | **Descrição** |
|---|---|
| ! | Espaços em branco não serão considerados no texto |
| > | Todos os caracteres seguintes serão maiúsculos até que apareça o caractere < |
| < | Todos os caracteres seguintes serão minúsculos até que apareça o caractere > |
| \ | Indica um caractere literal |
| l | Somente caractere alfabético |
| L | Obrigatoriamente um caractere alfabético |
| a | Somente caractere alfanumérico |
| A | Obrigatoriamente caractere alfanumérico |
| 9 | Somente caractere numérico |
| 0 | Obrigatoriamente caractere numérico |
| c | Permite um caractere |
| C | Obrigatoriamente um caractere |
| # | Permite um caractere numérico ou sinal de mais ou de menos, mas não os requer. |
| : | Separador de horas, minutos e segundos |
| / | Separador de dias, meses e anos |

**Tmemo** - Permite entrada de dados texto em múltiplas linhas. Contém propriedades e métodos do TEdit.

| **Propriedade** | **Descrição** |
|---|---|
| Lines | Propriedade do tipo TStrings que armazena as linhas de texto do componente |
| WantReturns | Define se a tecla ENTER será tratada como quebra de linha |
| WantTabs | Define se a tecla TAB será tratada como espaço de tabulação |
| ScrollBar | Define as barras de rolagem |

**Tstrings -** Muitos componentes, como o TMemo, possuem propriedades do Tipo TStrings, essa classe permite armazenar e manipular uma lista de Strings. Toda propriedade do tipo TStrings permite acesso indexado aos itens da lista.

| **Propriedade** | **Descrição** |
|---|---|
| Count | Número de strings |
| Text | Conteúdo do memo na Forma de uma única string |
| **Método** | **Descrição** |

| | |
|---|---|
| Add | Adiciona uma nova string no final da lista |
| Insert | Insere uma nova string numa posição especificada |
| Move | Move uma string de um lugar para outro |
| Delete | Apaga uma string |
| Clear | Apaga toda a lista |
| IndexOf | Retorna o índice do item e - 1 caso não encontre |
| LoadFromFile | Carrega texto de um arquivo |
| SaveToFile | Salva texto para um arquivo |

**TcheckBox -** Utilizado para obter informações de checagem.

| **Propriedade** | **Descrição** |
|---|---|
| AllowGrayed | Determina se o checkbox terá três possibilidades de estado |
| Checked | Determina se o checkbox está marcado |
| State | Estado atual do checkbox |

**TradioButton -** Usado em grupo, pode ser utilizado para obter inFormações lógicas mutuamente exclusivas, mas é recomendado usar o RadioGroup em vez de RadioButtons.

**TradioGroup -** Componente que agrupa e controla RadioButtons automaticamente.

| **Propriedade** | **Descrição** |
|---|---|
| Columns | Número de colunas de RadioButtons |
| Items | Lista de strings com os itens do RadioGroup, cada item da lista representa um RadioButton |
| ItemIndex | Item selecionado, iniciando em 0 |

**Tpanel** - Componente Container utilizado para agrupar componentes em um painel.

| **Propriedade** | **Descrição** |
|---|---|
| BevelInner | Estilo da moldura interna do painel |
| BevelOuter | Estilo da moldura externa do painel |
| BevelWidth | Largura das molduras |
| BorderStyle | Estilo da Borda |
| BorderWidth | Largura da borda, distância entre as molduras interna e externa |

**TscrollBox** - Container com barras de rolagem automáticas**.**

**TgroupBox** - Componente container com um título e borda 3D.

**Tbevel** - Moldura ou linha com aparência 3D.

| **Propriedade** | **Descrição** |
|---|---|
| Shape | Tipo de moldura a ser desenhada |
| Style | Define alto ou baixo relevo para a linha |

**TlistBox** - Utilizado para exibir opções em uma lista.

| **Propriedade** | **Descrição** |
|---|---|
| Columns | Número de colunas de texto da lista |
| MultiSelect | Define se será permitida a seleção de múltiplos itens |
| ExtendedSelect | Define se a seleção poderá ser estendida pelo uso das teclas Shift e Ctrl |
| IntegralHeight | Define se os itens poderão aparecer parcialmente ou somente por completo |
| Items | Lista de strings com os itens da lista |
| ItemIndex | Índice do item selecionado, começando em 0 |
| Selected | De acordo com o índice indica se um item em particular esta selecionado |

| | |
|---|---|
| SelCount | Indica quantos itens estão selecionado |
| Sorted | Define se os itens aparecerão ordenados |

**TcomboBox -** Caixa combinada com lista suspensa.

| **Propriedade** | **Descrição** |
|---|---|
| Items | Lista de strings com os itens da lista |
| DropDownCount | Número de itens visíveis da lista suspensa |
| Style | Estilo do ComboBox, os principais estilos são csDropDown, csDropDownList, csSimple |

**Timage** - componente usado para exibir figuras.

| **Propriedade** | **Descrição** |
|---|---|
| Center | Determina de a figura será centralizada no componente |
| Picture | Figura a exibida, pode ser BMP, ICO, WMF ou EMF |
| Stretch | Define se o tamanho da figura deve ser ajustada ao do componente |

**Tpicture -** classe usada para guardar ícones, Bitmaps, meta arquivos do Windows ou gráficos definidos pelo usuário.

| **Método** | **Descrição** |
|---|---|
| LoadFromFile | Carrega figura de um arquivo |
| SaveToFile | Salva figura para um arquivo |

**TpageControl -** usado para criar controles com múltiplas páginas, que podem ser manipuladas, em tempo de projeto, através do menu de contexto. Cada página criada é um objeto do tipo TTabSheet.

| **Propriedade** | **Descrição** |
|---|---|
| ActivePage | Página ativa |
| MultiLine | Define múltiplas linhas de guias de páginas |
| TabHeigth | Altura das guias |
| TabWidth | Largura das guias |
| **Evento** | **Descrição** |
| OnChange | Após uma mudança de página |
| OnChanging | Permite a validação de uma mudança de página |
| **Método** | **Descrição** |
| FindNextPage | Retorna a próxima página |
| SelectNextPage | Seleciona a próxima página |

**TtabSheet** - página de um PageControl.

| **Propriedade** | **Descrição** |
|---|---|
| PageIndex | Ordem da página |
| TabVisible | Define se a aba da página é visível |

**Tshape -** gráfico de uma Forma geométrica.

| **Propriedade** | **Descrição** |
|---|---|
| Brush | Preenchimento da figura, objeto do tipo TBrush |
| Pen | Tipo da linha, objeto do tipo TPen |
| Shape | Forma geométrica |

**Ttimer -** permite a execução de um evento a cada intervalo de tempo.

| **Propriedade** | **Descrição** |
|---|---|
| Interval | Tempo em milissegundos quando o componente irá disparar o evento OnTimer |
| **Evento** | **Descrição** |
| OnTimer | Chamado a cada ciclo de tempo determinado em Interval |

**TstatusBar -** utilizado para criar barras de status para exibir inFormações.

| Propriedade | Descrição |
|---|---|
| SimplePanel | Indica se haverá apenas um panel |
| SimpleText | Texto exibido caso SimplePanel seja True |
| SizeGrip | Define se a alça de redimensionamento padrão deve ser mostrada |
| Panels | Propriedade do tipo TStatusPanels, com os painéis do StatusBar |

**TstatusPanels -** lista de panels de um StatusBar.

| Propriedade | Descrição |
|---|---|
| Count | Número de panels |
| Items | Lista de panels, cada panel é um objeto do tipo TStatusPanel |
| **Método** | Descrição |
| Add | Adiciona um novo panel à lista |

**TstatusPanel-** panel de um StatusBar.

| Propriedade | Descrição |
|---|---|
| Text | Texto do panel |
| Width | Largura em pixels |
| Bevel | Moldura do panel |
| Alignment | Alinhamento do texto de um panel |

## *4.7 Caixas de Diálogo*

Grupo de caixas de diálogo comuns do Windows.

| Método | Descrição |
|---|---|
| Execute | Mostra a caixa de diálogo e retorna True caso o usuário clique em Ok |

**TOpenDialog / TsaveDialog -** caixas de diálogo para abrir e salvar arquivos.

| Propriedade | Descrição |
|---|---|
| FileName | Nome do arquivo |
| DefaultExt | Extensão padrão para os arquivos |
| Filter | Filtro, com os tipos de arquivos que serão abertos ou salvos |
| FilterIndex | Índice do filtro default |
| InitialDir | Pasta inicial |
| Title | Título da janela |
| Options | Define características gerais do diálogo |

**TfontDialog -** caixa de diálogo de escolha de fonte.

| Propriedade | Descrição |
|---|---|
| Device | Define se deve utilizar fontes para tela, impressora ou ambos |
| MinFontSize | Tamanho mínimo da fonte |
| MaxFontSize | Tamanho máximo da fonte |
| Options | Define características das fontes |
| **Evento** | **Descrição** |
| OnApply | Ocorre após o usuário pressionar o botão Aplicar, antes da janela fechar |

## *4.8 Menus*

No Delphi os menus serão desenhados no Menu Designer, que pode ser acessado no menu de contexto de qualquer componente de menu.

**TmainMenu -** menu principal de um Form.

| Propriedade | Descrição |
|---|---|
| Items | Itens de menu, essa propriedade guarda todas as alterações feitas no Menu Designer |

**TpopUpMenu** - Menu de contexto de um componente. Cada componente tem uma propriedade PopUpMenu, que indica seu menu de contexto.

**TmenuItem** - item de menu.

| Propriedade | Descrição |
|---|---|
| Checked | Indica se o item está marcado ou não |
| GroupIndex | Índice do grupo do item, semelhante ao SpeedButton |
| RadioGroup | Indica se o item pode ser mutuamente exclusivo com outros itens do mesmo grupo |
| ShortCut | Tecla de atalho do item |

## *4.9 Classes Não Visuais*

**Tapplication -** todo programa tem um objeto global nomeado Application, do tipo TApplication, esse objeto representa a aplicação para o Windows.

| Propriedade | Descrição |
|---|---|
| ExeName | Caminho e nome do arquivo executável |
| MainForm | Form principal da aplicação |
| Hint | Hint recebido pela aplicação |
| Title | Título da aplicação |
| HelpFile | Caminho e nome do arquivo help |
| **Evento** | **Descrição** |
| OnHint | Quando um hint é recebido pela aplicação |
| OnException | Quando ocorre uma exceção |
| OnHelp | Quando acontece uma solicitação de help |
| **Método** | **Descrição** |
| MessageBox | Apresenta um quadro de mensagem |
| Run | Executa a aplicação |
| Terminate | Finaliza a aplicação normalmente |

**Quadros de Mensagem -** o método Application.MessageBox mostra quadros de mensagem com chamadas a funções da API do Windows. Os flags de mensagem mais usados e os valores de retorno desse método são mostrados abaixo.

| Flag | Item Mostrado |
|---|---|
| MB_ABORTRETRYIGNORE | Botões de Abortar, Repetir e Ignorar |
| MB_ICONERROR | Ícone de erro |
| MB_ICONEXCLAMATION | Ícone com ponto de exclamação |
| MB_ICONINFORMATION | Ícone com letra i, usada para mostrar inFormações |
| MB_ICONQUESTION | Ícone de pergunta |
| MB_OK | Botão de Ok |
| MB_OKCANCEL | Botões de Ok e Cancelar |
| MB_RETRYCANCEL | Botões de Repetir e Cancelar |
| MB_SYSTEMMODAL | O Windows só poderá ser usado quando o quadro for fechado |
| MB_YESNO | Botões de Sim e Não |
| MB_YESNOCANCEL | Botões de Sim, Não e Cancelar |
| **Valor de Retorno** | **Botão Escolhido** |
| IDABORT | Abortar |
| IDCANCEL | Cancelar |
| IDIGNORE | Ignorar |

| | |
|---|---|
| IDNO | Não |
| IDOK | Ok |
| IDRETRY | Repetir |
| IDYES | Sim |

Esses quadros são usados quando se deseja uma resposta simples do usuário, principalmente numa confirmação ou pergunta para o usuário, como o código abaixo, usado no evento OnCloseQuery do Form principal.

```
if Application.MessageBox('Deseja fechar a aplicação?', 'Sair do sistema',
MB_ICONQUESTION +
  MB_YESNO) = IDNO then
  CanClose := False;
```

**Tscreen** - o Delphi automaticamente cria a variável Screen do tipo Tscreen, essa variável guarda características do vídeo, como mostrado abaixo.

| **Propriedade** | **Descrição** |
|---|---|
| ActiveForm | Form com o foco |
| FormCount | Número de Forms no vídeo |
| Cursor | Cursor do mouse |
| Forms | Lista dos Forms disponíveis |
| Fonts | Lista de Fontes de tela disponíveis |
| PixelsPerInch | Número de pixels por polegada da Fonte usada pelo sistema |
| Height | Altura da tela em pixels |
| Width | Largura da tela em pixels |

**Tprinter** - na Unit Printers é declarado um objeto do tipo TPrinter nomeado Printer que encapsula toda a interface de impressão do Windows e pode ser usado para imprimir diretamente, sem usar componentes de relatório, como o QuickReport.

| **Propriedade** | **Descrição** |
|---|---|
| Canvas | Superfície de desenho, do tipo TCanvas, onde será desenhada a imagem a ser impressa |
| Printers | Lista de impressoras instaladas |
| Orientation | Retrato ou Paisagem |
| PageHeight | Altura da página |
| PageWidth | Largura da página |
| PageNumber | Página atual |
| **Método** | **Descrição** |
| BeginDoc | Inicia o processo de desenho |
| EndDoc | Finaliza o processo de desenho e envia a imagem do Canvas para a impressora |
| Abort | Aborta a impressão |

**Tcanvas -** um objeto da classe TCanvas é uma superfície de desenho, onde podem ser usados vários métodos de plotagem gráfica. Todos os controles visíveis possuem uma propriedade Canvas, do tipo TCanvas, que geralmente é usada nos Forms e no objeto Printer.

| **Propriedade** | **Descrição** |
|---|---|
| Brush | Padrão de preenchimento, propriedade do tipo TBrush |
| Pen | Estilo de linha, propriedade do tipo TPen |
| Font | Fonte usada nas plotagens de texto |
| **Método** | **Descrição** |
| TextOut | Desenha texto na superfície |
| Ellipse | Desenha uma elipse |
| Polygon | Desenha um polígono |
| Rectangle | Desenha um retângulo |

**Tlist** - estrutura de dados polimórfica que pode gerenciar uma lista de objetos de qualquer classe e possui métodos semelhantes aos de TStrings.

**TstringList** - lista de strings descendente de TStrings usada para manter listas de strings independentes de qualquer componente.

**Tregistry -** interface com a API de manipulação do Registry do Windows, banco de dados de configuração do sistema.

# 5 BANCOS DE DADOS

## *5.1 Conceitos Importantes*

O gerenciamento de bancos de dados é essencial para o desenvolvimento comercial, em particular para sistemas de informação que possuem grande volume de informações para serem proessadas. Para se criar um banco de dados eficiente é necessário o conhecimento prévio de modelagem de bancos de dados relacionais. Conceitos como banco de dados, tabelas, campos, registros, índices, chaves, relacionamentos, normalização, dentre outros são pré-requisitos básicos para o desenvolvimento desse conteúdo.

## *5.2 Borland Database Engine*

A BDE (*Borland Database Engine*) fornece a capacidade de acesso padronizado a banco de dados para Delphi, C++ Builder e outros ambientes de programação da Borland, oferecendo um grande conjunto de funções para auxiliar no desenvolvimento de aplicações Desktop e Cliente/Servidor.

Os controladores da BDE podem ser usados para acessar bases de dados dBase, Paradox, Access, FoxPro, Interbase, Oracle, Sybase e MS-SQL Server, DB2, Informix, além de um controlador de acesso a arquivos texto. Você também pode utilizar fontes de dados ODBC, podendo acessar qualquer base de dados compatível.

As funções que compõe uma API da BDE são usadas internamente pelos componentes de acesso a dados do Delphi e muito raramente você teria que usá-las diretamente, mas isso é totalmente possível. A referência completa das funções da BDE, com exemplos em Delphi, está no BDE API Help na pasta do Delphi no Menu Iniciar.

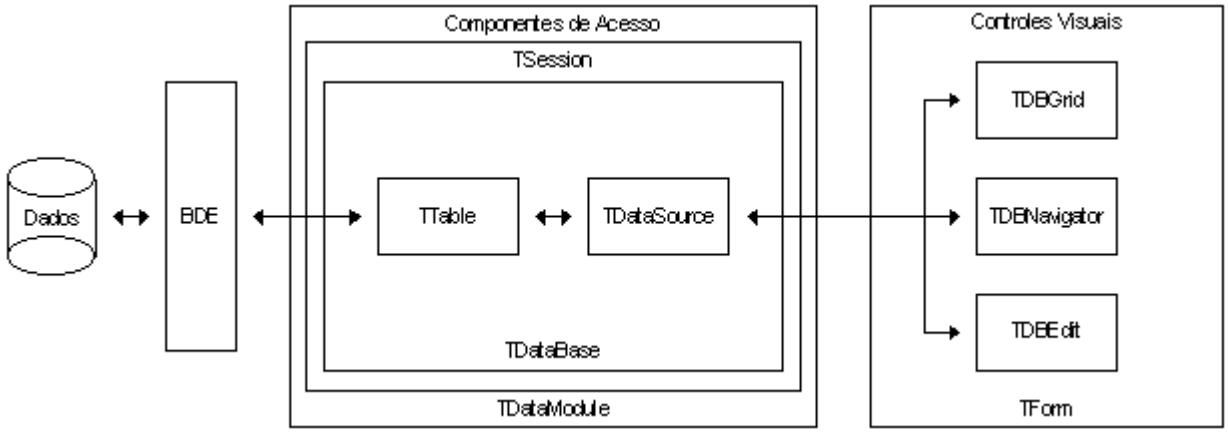
Para configurar o acesso a um banco de dados, você tem várias opções, criar um Alias, usar o componente TDatabase ou os dois juntos.

### Fonte de Dados

Uma Fonte de Dados (ou Alias) é um nome lógico, um atalho para um banco de dados. Todo o trabalho do Delphi com um banco de dados pode ser feito baseado na Fonte de Dados, de forma que para mudar de banco de dados, só é necessário mudar o Fonte de Dados. Para criar um Fonte de Dados você pode usar *Database Explorer*, o *BDE Administrator* ou o próprio *Database Desktop*.

## *5.3 Arquitetura de Acesso*

A acesso e manipulação de um banco de dados por um programa Delphi é realizada como mostrado a seguir, note que a aplicação não acessa os dados diretamente, mas usa sempre a BDE.




Assim, para uma aplicação de bancos de dados funcionar, é preciso que a BDE esteja instalada na máquina, não bastando apenas o arquivo executável.

## *5.4 Data Access*

Controles usados para fazer conexão com banco de dados.

| Componente | Utilidade |
|---|---|
| **TDataBase** | Instala uma conexão permanente com um banco de dados, especialmente um banco de dados remoto requerendo um login do usuário e uma senha. |
| **TDataModule** | É uma Unit que pode ser conter as classes TTable e TQuery para evitar repetições destes componentes dentro da aplicação. |
| **TTable** | Recupera dados de um tabela de banco de dados via o BDE e abastece ele para um ou mais componentes de controle de dados através do componente TDataSource. Envia dados recebidos de um componente para uma banco de dados via o BDE. |
| **TQuery** | Usa estruturas em SQL para recuperar dados de uma tabela de banco de dados via o BDE, e abastece ele para um ou mais componentes de controle de dados através de um TDataSouce, ou usa estruturas em SQL para enviar dados de um componente para um banco de dados via o BDE. |
| **TUpdateSQL** | Acoplado ao TQuery e oferece 3 strings para comando SQL (inserção, alteração e deleção. |
| **TStoreProc** | Habilita uma aplicação para acessar procedimentos armazenados em um servidor. Envia dados recebidos de um componente para um banco de dados via o BDE. |
| **TSession** | Configurações do BDE no instante em que foi aberto o aplicativo. |
| **TBatchMove** | Copia a estrutura de uma tabela ou seus dados. Pode ser usado para mover tabelas inteiras de uma formato de banco de dados para outro. |

**Tdatabase –** esse componente permite a manipulação de um banco de dados, através de um Alias da BDE ou a criação de um Alias local, somente visível dentro da aplicação, esse componente também permite o gerenciamento de transações, garantindo uma integridade maior no projeto. Por essas e outras razões o uso do componente Database é altamente recomendado como opção para criação de Aliases.

| Propriedades | Descrição |
|---|---|
| AliasName | Nome do Alias do banco de dados, usado quando você criar um Alias da BDE |
| Connected | Define se a conexão com o banco de dados está ativa |
| DatabaseName | Nome do Alias local a ser usado pelos outros componentes do Delphi |
| DataSetCount | Número de DataSets (Tabelas) abertos no banco de dados |
| DataSets | Lista com os DataSets abertos |
| DriverName | Driver usado para criar um Alias local, automaticamente cancela a propriedade AliasName |
| InTransaction | Define se o Database está em transação |
| KeepConnection | Define se a conexão com o banco de dados será mantida, mesmo sem DataSets abertos |
| LoginPrompt | Define se será mostrado o quadro de login padrão da BDE |
| Params | Parâmetros do banco de dados, com itens semelhantes à seção Definition do Database |

| | |
|---|---|
| | Explorer |
| TransIsolation | Nível de isolamento da transação, define como uma transação irá enxergar outra |
| **Métodos** | **Descrição** |
| Close | Encerra a conexão com o banco de dados, todos os DataSets serão fechados |
| CloseDataSets | Fecha todos os DataSets abertos, mas a conexão não é encerrada |
| Commit | Grava alterações feitas durante a transação |
| Open | Abre a conexão com o banco de dados |
| Rollback | Anula todas as alterações feitas durante a transação |
| StartTransaction | Inicia uma transação |
| **Eventos** | **Descrição** |
| OnLogin | Evento usado quando você quiser escrever seu próprio método de conexão com o banco de dados |

Para acessar uma base de dados Access, você poderia usar os valores mostrados na descrição textual a seguir.

```
AliasName = 'Northwind'
DatabaseName = 'Dados'
LoginPrompt = False
KeepConnection = True
Params.Strings = (
  'DATABASE NAME=C:\Meus Documentos\NorthWind.mdb'
  'USER NAME=paulo'
  'OPEN MODE=READ/WRITE'
  'LANGDRIVER=intl850'
  'PASSWORD=elvis')
```

Para ajudar a preencher os parâmetros de um Database, clique duas vezes sobre o componente e clique em Defaults, todos os parâmetros defaults serão apresentados.

Para acessar uma base Paradox, use as propriedades abaixo, note que para o Paradox, a única informação realmente significante é o Path, a pasta onde estão as tabelas.

```
AliasName = 'DBDEMOS'
DatabaseName = 'Dados'
LoginPrompt = False
KeepConnection = True
Params.Strings = (
  'PATH=d:\Borland\Delphi 3\Demos\Data'
  'ENABLE BCD=FALSE'
  'DEFAULT DRIVER=PARADOX')
```

Após a criação do Alias da BDE ou do Alias local, usando o componente TDatabase, o banco de dados está configurado e pronto para ser usado.

**TdataModule -** um DataModule é como se fosse um Form invisível, onde iremos inserir os componentes de acesso a dados, como o Table e o Datasource. Por serem também classes, os DataModules permitem a fácil implementação de modelos de objetos, permitindo herança, criação de métodos, dentre outros aspectos. Para inserir um DataModule em um projeto, escolha New DataModule do menu File. Os DataModules não gastam recursos do sistema, servem apenas para conter os componentes de acesso a dados e criar, assim, uma classe persistente.

**Ttable -** Componente usado para acessar uma tabela em um banco de dados. Esse componente é o mais importante quando acessamos bases de dados Desktop. Muitas dos itens mostrados abaixo estão definidos na classe TDataSet, ancestral do TTable.

| **Propriedades** | **Descrição** |
|---|---|
| Active | Define se a tabela esta aberta ou fechada |
| BOF | Informa se está no início da tabela |
| CanModify | Define se a aplicação pode inserir, deletar ou alterar registros |

| | |
|---|---|
| DatabaseName | Nome do banco de dados onde está a tabela, deve ser escolhido um Alias, que pode ser local |
| EOF | Informa se está no fim da tabela |
| Exclusive | Define se a tabela pode ser compartilhada por outro usuário |
| FieldCount | Número de campos da tabela |
| FieldDefs | Lista com a Definição dos campos da tabela |
| Fields | Lista de objetos do tipo TField, que representam os campos da tabela |
| Filter | String com uma condição de filtragem |
| Filtered | Define se a tabela é filtrada |
| IndexFieldNames | Nome dos campo de índice, usados para ordenar os registros da tabela |
| IndexName | Nome do índice atual, vazia quando o índice for a chave primária |
| IndexDefs | Lista com a definição dos índices |
| MasterFields | Campos usados no relacionamento com a tabela mestre |
| MasterSource | DataSource da tabela mestre em uma relação Mestre/Detalhe |
| Modified | Define se o registro atual foi modificado |
| ReadOnly | Define se a tabela é somente para leitura |
| RecNo | Número do registro atual |
| RecordCount | Número de registros |
| State | Estado da tabela |
| TableName | Nome da tabela |
| TableType | Tipo da tabela |
| **Método** | **Descrição** |
| AddIndex | Cria um novo índice, a tabela deve ser exclusiva |
| Append | Entra em modo de inserção e, ao gravar, o registro será colocado no fim do arquivo |
| AppendRecord | Insere um registro no final do arquivo através de código |
| Cancel | Cancela as alterações feitas no registro atual |
| Close | Fecha a tabela |
| CreateTable | Cria uma tabela, depende de FieldDefs e IndexDefs |
| Delete | Exclui o registro corrente |
| DeleteIndex | Exclui um índice |
| DeleteTable | Exclui a tabela |
| DisableControls | Desabilita a atualização dos controles visuais |
| Edit | Permite a alteração dos campos do registro atual |
| EmptyTable | Apaga todos os registro da tabela, para isso a tabela não pode esta sendo compartilhada |
| EnableControls | Habilita os controles visuais |
| FieldByName | Acessa um campo, do tipo TField, pelo nome |
| FindKey | Procura o registro com os valores exatos aos dos parâmetros nos campos do índice atual |
| FindNearest | Procura o registro com os valores mais aproximados aos dos parâmetros nos índices |
| First | Move para o primeiro registro |
| Insert | Entra em modo de inserção de um novo registro na posição atual |
| InsertRecord | Adiciona um novo registro, já com os dados, na posição atual |
| IsEmpty | Define se a tabela está vazia |
| Last | Move para o último registro |
| Locate | Procura um registro, usando ou não índices, de acordo com a disponibilidade |
| LockTable | Trava a tabela |
| Lookup | Procura um registro e retorna valores dos campos deste |
| MoveBy | Move um número específico de registros |
| Next | Move para o próximo registro |
| Open | Abre a tabela |
| Post | Grava as alterações no registro atual |
| Prior | Move para o primeiro registro |
| Refresh | Atualiza a tabela com os dados já gravados |
| RenameTable | Renomeia a tabela |
| UnlockTable | Destrava a tabela |
| **Evento** | **Descrição** |
| AfterCancel | Após do método Cancel |

| | |
|---|---|
| AfterClose | Após o fechamento da tabela |
| AfterDelete | Após do método Delete |
| AfterEdit | Após do método Edit |
| AfterInsert | Após do método Insert |
| AfterOpen | Após do método Open |
| AfterPost | Após do método Post |
| AfterScroll | Após mudar de registro |
| BeforeCancel | Antes do método Cancel |
| BeforeClose | Antes do fechamento da tabela |
| BeforeDelete | Antes do método Delete |
| BeforeEdit | Antes do método Edit |
| BeforeInsert | Antes do método Insert |
| BeforeOpen | Antes do método Open |
| BeforePost | Antes do método Post |
| BeforeScroll | Antes de mudar o registro |
| OnCalcFields | Evento usado para calcular os valores dos campos calculados |
| OnDeleteError | Quando ocorre um erro ao chamar o método Delete |
| OnEditError | Quando ocorre um erro ao chamar o método Edit |
| OnFilterRecord | Evento usado com filtragem variável |
| OnNewRecord | Quando a tabela entra em modo de inserção, não deixa Modified igual a True |
| OnPostError | Quando ocorre um erro ao chamar o método Post |

**Usando DataSets -** as classes de componentes TTable e TQuery são descendentes de TDataset através do TDBDataSet. Esta classe de componente herda uma parte de propriedades, métodos e eventos. Os estados do DataSet:

| **Estado** | **Descrição** |
|---|---|
| dsInactive | O Dataset esta fechado. |
| dsBrowse | Estado de Espera. O estado default quando um dataset é aberto. Registros pode ser visualizados mas não mudados ou inseridos. |
| dsEdit | Habilita o a linha corrente para ser editada. |
| dsInsert | Habilita uma nova linha para ser inserida. Uma chamada para o Post inserir a nova linha. |
| dsSetKey | Habilita FindKey, GotoKey, and GoToNearest para procurar valores nas tabelas do banco de dados. Estes métodos somente pertencem para o componente TTable. Para TQuery, a procura é feita com a syntax do SQL. |
| dsCalcFields | Modo quando os OnCalcFields é executado; previne qualquer mudança para outros campos ou campos calculados. Raramente é usado explicitamente. |

**Abrindo e Fechado DataSets** - antes que uma aplicação possa acessar dados através de um dataset, o dataset deve ser aberto. Existem dois meios para abrir o dataset:

Ativando a propriedade Active para *True*, isto pode ser feito através do Object Inspector ou programavelmente em tempo de execução.

```
Table1.Active := True;
```

Chamando o método do Dataset *Open* em tempo de execução.

```
Table1.open;
```

Para fechar o processo e semelhante só muda o a propriedade para *False* e o evento para ***Close;***

**Lendo Valores do Campo** - existem algumas maneiras de ler dados de um dataset:

```
Edit1.text := Table1Nome_Clie.asstring;
Edit1.text := Table1.Fields[1].asstring;
Edit1.text := Table1.FieldByName('Nome_Clie').asstring;
```

Para associar outros tipos de campos que não texto a uma caixa de edição (que só aceita valores do tipo string), devemos utilizar propriedades de conversão do componente TField: AsBoolean, AsDataTime, AsFloat (Real), AsInteger, AsString.

Para atribuir valores para o dataset o procedimento é o mesmo só que no sentido inverso, desde que que a tabela esteja em modo de edição ou inserção.

**Inserindo Registros** - para inserir registros em código você pode usar os métodos AppendRecord e InsertRecord, caso você não precise de algum campo, mesmo assim ele deve ser informado com o valor *Null*.

```
Table1.AppendRecord([Null, EdtDescricao.Text, EdtPreco.Text]);
```

Outra forma de inserir um registro:

```
Table1.Insert;
Table1Data.Value := Date;
Table1Hora.Value := Time;
Table1.Post;
```

**Alterando Registros -** para alterar registros em código, colocamos a tabela em modo de edição, alteramos o valor dos campos e gravamos as alterações, se for necessário.

```
Table1.Edit;
Table1Data.Value := Date;
Table1Hora.Value := Time;
Table1.Post;
```

**Verificando Alterações** - onde for necessário a verificação de alterações feitas em uma Tabela, por exemplo no evento OnClose de um Form de manutenção, você pode usar a propriedade Modified, como mostrado no exemplo abaixo.

```
if Table1.Modified then
   if Application.MessageBox('Gravar alterações?', 'Dados Alterados',
   MB_ICONQUESTION + MB_YESNO) = IDYES then
      Table1.Post
   else
      Table1.Cancel;
```

**Valores Default -** caso você queira especificar valores Default para os campos de uma tabela, use o evento OnNewRecord, pois nesse evento o registro não é marcado como modificado.

```
Table1Data.Value := Date;
```

**Marcando Dados -** os métodos para marcar são

- **GetBookmark** - Pega a posição do registro e coloca em uma variável do tipo TBookmark;
- **GotoBookmark** - Leva o apontador para uma variavel do tipo TBookmark;
- **FreeBookmark** - Desaloca uma variavel do tipo TBookmark;

```
procedure DoSomething;
var
   Marca: TBookmark;
begin
   Marca  :=  Table1.GetBookmark; {Marca a posição}
   Table1.DisableControls; {Desabilita o controle de dados}
   try
       Table1.First;
       while not Table1.EOF do
            begin
               {Faz alguma coisa}
               Table1.Next;
            end;
```

```
    finally
       Table1.GotoBookmark(Marca);
       Table1.EnableControls;
       Table1.FreeBookmark(Marca); {Desaloca a variavel}
    end;
end;
```

**Indexação -** a indexação é usada para ordenar os registros da tabela.

| **Método** | **Descrição** |
|---|---|
| GetIndexNames | Retorna uma lista de nomes disponiveis de índeces para uma tabela do banco de dados. |
| IndexFieldCount | Número de Campos do Índice. |
| IndexFields | Vetor de nomes de campos usados no índice |
| IndexName | Para usar o índice primário de uma tabela Paradox. |
| IndexFieldNames | Para muda o índice para qualquer campo. |

As propriedades IndexName e IndexFieldNames são mutualmente exclusivas. Colocando uma propriedade, limpa o valor da outra.

Para ordenar você deve escolher os campos pelos quais você quer ordenar na propriedade IndexFieldNames, inclusive em código, como mostrado abaixo, todos campos devem ser indexados e separados por ponto e vírgula.

```
Table1.IndexFieldNames := 'Nomcli';
Table1.IndexFieldNames := 'Data, Vendedor';
```

**Percorrendo uma Tabela -** utilize um código semelhante ao mostrado abaixo para percorrer uma tabela do início ao fim.

```
Table1.DisableControls;
Total := 0;
Table1.First;
while not Table1.EOF do
begin
   Total := Total + Table1Valor.Value;
   Table1.Next;
end;
Table1.EnableControls;
```

**Procurando dados em uma Tabela** - os métodos GoToKey e GoToNearest habilita uma aplicação para procurar uma chave em uma tabela. SetKey coloca a tabela em modo de procura. No estado SetKey, atribui-se valores para a procura em um campo indexado. GoToKey então move o cursor para a primeira linha da tabela que encontrou este valor. Como GoToKey é uma função ela retorna True ou False se teve sucesso na procura. O GoToNearest é uma função que leva para o valor mais parecido encontrado.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.SetKey; {Primeiro campo é a chave}
  Table1.Fields[0].AsString := Edit1.Text;  (* Table1.Fieldbyname('Cidade')
:= Edit1.Text *)
  Table1.GoToKey;
end;
```

Todo procedimento acima pode ser substituído usando a função *Find*

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.FindKey([Edit1.text]);
end;
```

Exemplo de outra forma de procura em uma tabela.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.SetKey; {Primeiro campo é a chave}
  Table1.Fieldbyname('Cidade') := Edit1.Text;
  Table1.GoToKey;
end;
```

Procura em um indíce secundário.

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  Table1.indexname := 'Nome_Clie';
  Table1.open;
  Table1.setkey;
  Table1Nome_clie.asstring := Edit1.text;
  Table1.gotonearest;
end;
```

Procura em um indíce primário ou secundário.

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  Table1.indexfieldnames := 'Codi_Clie';
  Table1.open;
  Table1.setkey;
  Table1Nome_clie.asstring := Edit1.text;
  Table1.gotonearest;
end;
```

**Localizando Registros -** para localizar registros você pode usar vários métodos, mas o melhor deles é o Locate, no exemplo abaixo é feita uma pesquisa exata.

```
if not Table1.Locate('CodCli', Edt.Text, []) then
  ShowMessage('Cliente não encontrado.');
```

Você também pode fazer uma pesquisa parcial e/ou sem sensitividade de caso usando o terceiro parâmetro, que é um conjunto de opções.

```
Table1.Locate('Nome', Edt.Text, [loPartialKey, loCaseInsensitive]);
```

Se você precisar fazer uma pesquisa por mais de um campo, separe os nomes dos campos por ponto e vírgula e use a função VarArrayOf para criar um array com os valores que você quer procurar.

```
if not Table1.Locate('Vendedor;Data', VarArrayOf([EdtVendedor.Text, EdtData.Text]),
  [loCaseInsensitive]) then
  ShowMessage('O vendedor não realizou nenhuma venda nessa data');
```

Caso os campos pesquisados sejam indexados, a pesquisa será muito mais eficiente, senão será criado um filtro temporário da BDE para localizar os registros

**Filtros -** usando o *Filter*, você pode filtrar os registro de uma tabela usando uma expressão lógica, como nos exemplos abaixo. Para tornar um filtro ativo, basta colocar *Filtered* igual a True.

```
Table1.filter := 'Data = '20/04/1998''
Table1.filter := (Data = '20/04/1998') AND (Vendedor = 'Gilherme Augusto da
Fonseca')
Table1.filter := (Nome > 'A') AND (Nome < 'B')
```

Contudo, se a condição de filtragem for muito variável, é preferível usar um código como o mostrado abaixo no evento OnFilterRecord da Table, para fazer uma filtragem dinâmica, com a propriedade Filter vazia e Filtered igual a True.

```
Accept := TblData.Value = Date;
```

Ao filtrar uma tabela, a propriedade RecordCount da Table, só mostra o número de registros que satisfazem ao filtro, como se os outros registros nao existissem.

Você também pode filtar registros de uma tabela através de métodos específicos:

- **SetRangeStart**: Estabelece o limite inicial do filtro.
- **SetRangeEnd**: Estabelece o limite final do filtro.
- **ApplyRange**: Aplica o filtro à tabela.
- **CancelRange**: Cancela o filtro aplicado à tabela.
- **KeyExclusive**: Exclui a chame do indíce setado.

```
Table1.SetRangeStart;
Table1.FieldByName('Codigo'):=100;(*Table1Codigo.asinteger := 100 *)
Table1.KeyExclusive := False;
Table1.SetRangeEnd;
Table1.FieldByName('Codigo'):=200;(*Table1Codigo.asinteger := 200 *)
Table1. KeyExclusive := True;
Table1.ApplyRange;
```

Obterá um filtro dos registros de clientes com código entre 100 e 199.

**Sincronizando Tabelas -** o método GotoCurrent é um método que sincroniza dois componentes table ligados a uma mesma tabela em um banco de dados e usando o mesmo indíce.

```
Table1.GotoCurrent(Table2);
```

**Tfield -** a classe TField é usada como ancestral para todos as classes dos campos.

Geralmente iremos usar objetos de classes descendentes de TField, mas em todos eles podemos encontrar os itens mostrados abaixo.

| Propriedades | Descrição |
|---|---|
| Alignment | Alinhamento do texto do campo nos controles visuais |
| AsBoolean | Valor do campo convertido para Boolean |
| AsCurrency | Valor do campo convertido para Currency |
| AsDateTime | Valor do campo convertido para DataTime |
| AsFloat | Valor do campo convertido para Double |
| AsInteger | Valor do campo convertido para Integer |
| AsString | Valor do campo convertido para string |
| AsVariant | Valor do campo convertido para Variant |
| Calculated | Indica se o campo é calculado em tempo de execução |
| CanModify | Indica se um campo pode ser modificado |
| ConstraintErrorMessage | Mensagem de erro se a condição de CustomConstraint não for satisfeita |
| CustomConstraint | Condição de validação do campo |
| DataSet | DataSet onde está o campo |
| DataSize | Tamanho do campo, em Bytes |
| DataType | Propriedade do tipo TFieldType, que indica o tipo do campo |
| DefaultExpression | Expressão com valor Default do campo para novos registros |
| DisplayLabel | Título a ser exibido para o campo |
| DisplayText | Texto exibido nos controles visuais associados ao campo |
| DisplayWidth | Número de caracteres que deve ser usado para mostrar o campo no controles visuais |
| EditMask | Máscara de edição do campo |
| FieldKind | Propriedade do tipo TFieldKind que indica o tipo do campo, como Calculado ou Lookup |
| FieldName | Nome do campo na tabela |
| FieldNo | Posição física do campo na tabela |
| Index | Posição do campo nos controles visuais |
| IsIndexField | Indica se um campo é válido para ser usado como índice |

| | |
|---|---|
| IsNull | Indica se o campo está vazio |
| KeyFields | Campo chave da tabela no relacionamento com LookupDataSet, usado em campos Lookup |
| Lookup | Indica se um campo é Lookup |
| LookupCache | Define se será usado cache para campos Lookup |
| LookupDataSet | DataSet onde está definido o valor do campo Lookup |
| LookupKeyFields | Campo chave do relacionamento em LookupDataSet |
| LookupResultField | Valor do campo, que será mostrado nos controles visuais |
| ReadOnly | Define se um campo é somente para leitura |
| Required | Define se o campo é obrigatório |
| Size | Tamanho físico do campo |
| Text | Texto de edição do campo |
| Value | Acesso direto ao valor do campo |
| Visible | Define se um campo é visível |
| **Eventos** | **Descrição** |
| OnChange | Chamado quando o valor do campo é mudado |
| OnSetText | Chamado pelos controles visuais para atribuir o texto digitado pelo usuário ao campo |
| OnGetText | Chamado para formatar o texto de exibição do campo |
| OnValidate | Validação do valor atribuído ao campo, caso o valor não seja válido, gere uma exceção |
| **Método** | **Descrição** |
| Assign | Atribui um valor de um campo a outro, inclusive nulo |
| FocusControl | Seta o foco para o controle visual ligado ao campo nos Forms |
| Clear | Limpa o conteúdo do campo |

Estão listadas abaixo algumas classes que realmente iremos manipular no tratamento dos campos de uma tabela, são classes descendentes de TField.

| | | |
|---|---|---|
| TStringField | TBlobField | TTimeField |
| TSmallintField | TIntegerField | TBytesField |
| TFloatField | TWordField | TVarBytesField |
| TCurrencyField | TAutoIncField | TGraphicField |
| TBooleanField | TBCDField | TMemoField |
| TDateField | TDateTimeField | |

Em alguns desses campos você pode encontrar as propriedades mostradas abaixo, que não estão presentes em TField.

| **Propriedades** | **Descrição** |
|---|---|
| MaxValue | Valor máximo para o campo |
| MinValue | Valor mínimo para campo |
| DisplayFormat | Formato de apresentação do campo, como *,0.00" %"* ou *,0.##" Km"* |
| EditFormat | Formato de edição do campo |
| Currency | Define se um campo é monetário |
| DisplayValues | Usado com campos Boolean, define o texto para True e False, como *Sim;Não* |
| **Métodos** | **Descrição** |
| LoadFromFile | Carrega o conteúdo do campo de um arquivo |
| SaveToFile | Salva o conteúdo do campo para um arquivo |

**Fields Editor -** para criar objetos para os campos de uma tabela clique duas vezes no componente TTable ou escolha *Fields Editor* no seu menu de contexto, na janela do Fields Editor, clique com o botão direito do mouse e escolha Add, na janela Add Fields, escolha os campos que você vai querer usar e clique em Ok.

No Fields Editor podemos também remover os campos criados, alterar sua ordem de apresentação e usar suas propriedades e eventos no Object Inspector. Para cada campo é criado um objeto de um tipo descendente de TField, como TStringField, TIntegerField, TFloatField. As principais propriedades dos objetos TField estão listadas na tabela abaixo.

Se você não criar nenhum objeto TField, todos os campos da tabela estarão disponíveis, mas caso você crie algum, somente os campos que você criar estarão disponíveis.
Se você selecionar os campos no Fields Editor e arrastar para o Form, serão criados os controles visuais para esses campos, Label, DBEdit e outros, mas antes coloque a descrição dos campos na propriedade DisplayLabel.

**Conversão de Tipos -** a conversão de tipo de um campo pode ser feita através as propriedades tipo *As...*, como AsString.

```
Table1.AsString := EdtData.Text;
```

**Validação** - para validar os valores de um campo, você pode usar a propriedade CustomConstraint, por exemplo para garantir que a quantidade de um item seja maior que zero, use em CustomConstraint *Quantidade > 0*, e em CustomConstraint coloque a mensagem para o usuário caso a condição seja falsa. Outra forma, mais flexível, é usando o evento OnValidate, com um código como abaixo, onde é gerada uma exceção para cancelar a atribuição do valor ao campo.

```
if Table1Quantidade.Value <= 0 then
  raise Exception.Create('Quantidade deve ser maior que zero.');
```

**Formatação Personalizada -** caso queira fazer uma formatação personalizada do campo, pode usar os eventos OnGetText e OnSetText. Por exemplo, se tiver um campo *Estado*, e quiser que quando o valor do campo for *C* fosse mostrado *Casado* e *S*, *Solteiro*, no evento OnGetText use um código como o abaixo.

```
if TblEstado.Value = 'C' then
  Text := 'Casado'
else if TblEstado.Value = 'S' then
  Text := 'Solteiro';
```

Como controle visual para o usuário escolher o valor do campo, você poderia usar o DBComboBox, com *Solteiro* e *Casado* na propriedade *Items*, e no evento OnGetText do campo o código mostrado abaixo.

```
if Text = 'Casado' then
  TblEstado.Value := 'C'
else if Text := 'Solteiro' then
  TblEstado.Value = 'S';
```

**Campos Calculados -** para criar campos calculados, clique com o direito no Fields Editor e escolha *New Field*, no quadro NewField, digite o nome do campo, o nome do objeto será automaticamente informado, o tipo do campo, seu tamanho e escolha Calculated em Field type**.**

Para colocar um valor nesse campo usaremos o evento OnCalcFields do componente TTable, em nenhuma outra parte os valores desses campos podem ser alterados.

O código do evento OnCalcFields deve ser enxuto, pois este é chamado várias vezes durante a edição de um registro e um procedimento pesado pode comprometer a performance do sistema.

```
procedure TDtmAluno.TblCalcFields(DataSet: TDataSet);
begin
  if TblFaltas.Value > DtmTurma.TblMaxFaltas.Value then
    TblSituacao.Value := 'Evadido'
  else if TblNota.Value >= 7 then
    TblSituacao.Value := 'Aprovado'
  else
    TblSituacao.Value := 'Retido'
end;
```

**Campos Lookup** - para fazer um relacionamento, às vezes precisamos criar um campo de descrição, por exemplo em uma biblioteca, na tabela de empréstimos, temos o código do Livro, mas gostaríamos de mostrar o Título, esses campos são chamados de campos Lookup.

Para criar um campo Lookup, siga os passos abaixo, tomando como exemplo o caso do livro no empréstimo.

1. Abra o Fields Editor do Table desejado, Empréstimos
2. Clique com o direito e escolha New Field
3. No quadro New Field, escolha as propriedades do campo como descrito em campos calculados, mas em Field type, escolha Lookup
4. Em Key Fields escolha o campo da tabela que faz parte do relacionamento, CodLivro
5. DataSet é a tabela onde está a descrição, Livros
6. Em Lookup Keys, escolha o campo de DataSet que faz parte do relacionamento, CodLivro
7. Finalmente, escolha em Result field o campo de DataSet que vai ser mostrado para o usuário, Título

Essas opções correspondem a algumas propriedades do objeto TField gerado, que podem ser alteradas no Object Inspector, KeyFields, LookupDataSet, LookupKeyFields, LookupDataSet e LookupResultField.

Quando esses campo são exibidos em um DBGrid, por padrão é criado um botão de lookup que mostrará os valores da outra tabela uma lista. Para colocar esses campos em um Form, devemos usar o DBLookupComboBox, apenas com as propriedades padrão, DataSource e DataField, onde deve ser escolhido o campo Lookup, quando você arrastar o campo para o Form isso será feito automaticamente.

**TdataSource -** componente usado para fazer a ligação entre um DataSet e os componentes visuais.

| **Propriedade** | **Descrição** |
|---|---|
| AutoEdit | Define se a tabela entrará em modo de edição assim que o usuário digitar novos valores nos controles |
| DataSet | DataSet ao qual o TDataSource faz referência |
| **Evento** | **Descrição** |
| OnDataChange | Ocorre quando o DataSet é alterado, ao mudar de registro ou mudar os valores dos campos |
| OnStateChange | Ocorre quando o estado do DataSet é alterado |
| OnUpdateData | Ocorre antes de uma atualização |

**Tquery -**componente usado para enviar um comando SQL para um banco de dados ou uma instrução SELECT. Muitas dos itens mostrados abaixo estão definidos na classe TDataSet, ancestral do TTable.

| **Propriedades** | **Descrição** |
|---|---|
| Active | Define se a instrução SQL será executada ou não. Usado somente para instruções SELECT. |
| BOF | Informa se está no início da tabela |
| DatabaseName | Nome do banco de dados onde está a tabela, deve ser escolhido um Alias, que pode ser local |
| EOF | Informa se está no fim da tabela |
| FieldCount | Número de campos da tabela |
| FieldDefs | Lista com a Definição dos campos da tabela |
| Fields | Lista de objetos do tipo TField, que representam os campos da tabela |
| Filter | String com uma condição de filtragem |
| Filtered | Define se a tabela é filtrada |
| IndexFieldNames | Nome dos campo de índice, usados para ordenar os registros da tabela |
| IndexName | Nome do índice atual, vazia quando o índice for a chave primária |
| IndexDefs | Lista com a definição dos índices |
| Param | Lista de objetos do tipo TParam, que representam parâmentros dentro de uma instrução SQL |
| RecNo | Número do registro atual |
| RecordCount | Número de registros |
| SQL | Define a instrução SQL que será enviada para o Gerenciador |
| State | Estado da tabela |

| | |
|---|---|
| UpdateObject | Define um objeto UpdateSQL será utilizado |
| **Método** | **Descrição** |
| Close | Fecha instrução SQL. Usado somente para instruções SELECT. |
| ExecSQL | Executa um comando SQL. Usado para todas as instruções SQL menos SELECT. |
| FieldByName | Acessa um campo, do tipo TField, pelo nome |
| FindKey | Procura o registro com os valores exatos aos dos parâmetros nos campos do índice atual |
| FindNearest | Procura o registro com os valores mais aproximados aos dos parâmetros nos índices |
| First | Move para o primeiro registro |
| IsEmpty | Define se a tabela está vazia |
| ParamByName | Acessa um parâmetro, do tipo TParam, pelo nome |
| Last | Move para o último registro |
| Locate | Procura um registro, usando ou não índices, de acordo com a disponibilidade |
| Lookup | Procura um registro e retorna valores dos campos deste |
| MoveBy | Move um número específico de registros |
| Next | Move para o próximo registro |
| Open | Executa um comando SQL. Usado somente para instruções SELECT. |
| Post | Grava as alterações no registro atual |
| Prior | Move para o primeiro registro |
| Refresh | Atualiza a tabela com os dados já gravados |
| **Evento** | **Descrição** |
| AfterCancel | Após do método Cancel |
| AfterClose | Após o fechamento da tabela |
| AfterOpen | Após do método Open |
| AfterScroll | Após mudar de registro |
| BeforeClose | Antes do fechamento da tabela |
| BeforeDelete | Antes do método Delete |
| BeforeOpen | Antes do método Open |
| BeforeScroll | Antes de mudar o registro |
| OnCalcFields | Evento usado para calcular os valores dos campos calculados |
| OnFilterRecord | Evento usado com filtragem variável |

**Tparam** - lista de parâmentros para uma instrução SQL. Um parâmetro é como uma variável na instrução SQL. Para definir um parâmentro na instrução SQL basta colocar dois pontos ( : ) antes da variável.

```
Query1.SQL := 'Select * from Autor where id_autor = :pid_autor';
Query1.ParamByName( 'pid_autor' ).asinteger := 10;
Query1.Open;
```

**TUpdateSQL -** define 3 instruções SQL para um componente Query.

| **Propriedades** | **Descrição** |
|---|---|
| InsertSQL | Define a instrução SQL para inserir registros |
| DeleteSQL | Define a instrução SQL para apagar registros |
| ModifySQL | Define a instrução SQL para alterar registros |
| **Método** | **Descrição** |
| ExecSQL | Executa um comando SQL. No parâmentro (ukModify, ukInsert, ukDelete) defini qual instrução será usada |

## *5.5 Data Controls*

Controles usados na interface com o usuário. Todos esses componentes tem uma propriedade DataSource, que deve ter o DataSource do Table ao qual estão ligados.

| Componente | Utilidade |
|---|---|
| TDBGrid | Grade padrão de controle de dados que possibilita visualizar e editar dados de forma tabular, semelhante a uma planilha; faz uso extensivo das propriedades do TField (estabelecidos no Editor de Campos(Fields Editor)) para determinar a visibilidade de uma coluna, formato da visualização, ordem, etc. |
| TDBNavigator | Botões de navegação para controle de dados que move o apontador do registro corrente de uma tabela para o posterior ou anterior, inicia inserção ou modo de edição; confirma novos ou modificações de registros; cancela o modo de edição; e refresca(refaz) a tela para recuperar dados atualizados. |
| TDBText | Rótulo de controle de dados que pode mostrar um campo corrente do registro ativo. |
| TDBEdit | Caixa de edição de controle de dados que pode mostrar ou editar um campo corrente do registro ativo. |
| TDBCheckBox | Caixa de verificação de controle de dados que pode mostrar ou editar dados lógico de uma campo corrente do registro ativo. |
| TDBRadioGroup | Grupos de radio de controle de dados com botões de rádio que pode mostrar ou setar valores de colunas. |
| TDBMemo | Caixa memo de controle de dados que pode mostrar ou editar dados textos BLOB do registro corrente ativo. |
| TDBRichEdit | Caixa Rich Edit de controle de dados que pode mostrar ou editar dados textos BLOB do registro corrente ativo no formato (.RTF). |
| TDBImage | Caixa de Imagem de controle de dados que pode mostrar, cortar ou colar imagens BLOB do registro corrente ativo. |
| TDBListBox | Caixa de Lista de controle de dados que pode mostrar valores da coluna de uma tabela. |
| TDBComboBox | Caixa de Lista móvel de controle de dados que pode mostrar ou editar valores da coluna de uma tabela. |
| TDBLookupList | Caixa de Lista de controle de dados que mostrar valores mapeados através de outra tabela em tempo de compilação. |
| TDBLookupCombo | Caixa de Combo de controle de dados que mostrar valores mapeados através de outra tabela em tempo de compilação. |
| TDBChart | Usado para criar gráficos à partir de DataSet |

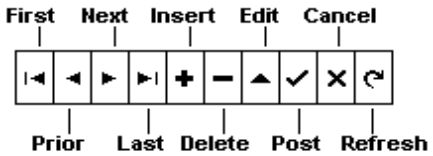
**TDBGrid -** este componente permite mostrar/editar os registros de um arquivo na forma de tabela. A vantagem deste tipo de saída de dados é que é possivel ver vários registros ao mesmo tempo. Mostra os registros de uma tabela em forma de grade, cada coluna é um campo e cada registro, uma linha.

| **Propriedades** | **Descrição** |
|---|---|
| Columns | Lista do tipo TDBGridColumns, com as colunas da Grid, cada item da lista é do tipo TColumn |
| DataSource | Componente DataSource que o controle está ligado |
| Fields | Lista de objetos TField mostrados nas colunas |
| Options | Set com as opções da Grid, como ConfirmDelete, MultiSelect, ColumnResize |
| SelectedField | Campo da coluna selecionada |
| SelectedIndex | Índice da coluna selecionada |
| SelectedRows | Lista do tipo TBookmarkList, com os registros selecionados em uma Grid com MultiSelect |
| TitleFont | Fonte do título das colunas |
| FixedColor | Cor Fixa, usada nas colunas e indicadores |
| **Eventos** | **Descrição** |
| OnCellClick | Ao clicar em uma célula da Grid |
| OnColEnter | Quando uma célula de alguma coluna da Grid recebe o foco |
| OnColExit | Quando uma célula de alguma coluna da Grid perde o foco |
| OnColumnMoved | Quando o usuário mover uma coluna |
| OnDrawDataCell | Evento usado para personalizar a forma de desenhar os dados que são apresentados na Grid |
| OnEditButtonClick | Ao clicar no botão de edição de uma célula, mostrado pela propriedade ButtonStyle da coluna |
| OnTitleClick | Ao clicar no título das colunas |

**Tcolumn** - item de uma lista *TDBGridColumns*, usada na propriedade *Columns* da Grid, objetos desse tipo representam uma coluna da Grid. Às vezes as propriedades definidas para o campo sobrepõem as propriedades

| **Propriedades** | **Descrição** |
|---|---|
| ButtonStyle | Botão mostrado ao editar as células da coluna |
| Field | Objeto TField ligado à coluna |
| FieldName | Nome do campo ligado à coluna |
| PickList | TStrings com os itens da lista DropDown usada nas células da coluna |
| Title | Propriedade do tipo TColumnTitle com as opções do título da coluna |

**TDBNavigator -** o DBNavigator permite que o usuário realize operações padrão de controle de dados. Cada um dos botões do DBNavigator chama um método do Componente Table ao qual está ligado.




Podemos personalizar o DBNavigator usando as suas propriedades e eventos, mas se quisermos mudar a figura dos botões teremos que editar diretamente o arquivo LIB\DBCTRLS.RES, na pasta do Delphi.

| **Propriedades** | **Descrição** |
|---|---|
| DataSource | Componente DataSource que o controle está ligado |
| VisibleButtons | Define os botões que serão visíveis |
| Hints | Hints exibidos pelos botões |
| ConfirmDelete | Define se será solicitado uma confirmação antes da exclusão |
| **Eventos** | **Descrição** |
| BeforeAction | Quando um botão do Navigator é pressionado, antes da ação ser executada |
| OnClick | Quando um botão do Navigator é pressionado, depois da ação ser executada |

**TDB*Text*** - usado para mostra os valores de campos.

| **Propriedades** | **Descrição** |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |

**TDBEdit -** usado para editar os valores dos campos.

| **Propriedades** | **Descrição** |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |

**TDBCheckBox** - usado em campos que podem receber apenas dois valores, como campos lógicos.

| **Propriedades** | **Descrição** |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |
| ValueChecked | Valor a ser armazenado quando está selecionado |
| ValueUnchecked | Valor a ser armazenado quando não está selecionado |

**TDBRadioGroup** - mostra algumas opções para o preenchimento de um campo.

| **Propriedades** | **Descrição** |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |
| Items | Título para cada botão de rádio |
| Values | Valor a ser armazenado para cada botão de rádio |

**TDBMemo** - usado para editar/visualizar campos com textos longos mas não usa formatação de caractere e de paragrafos. Geralmentes são armazenados nos SGBD em campos do tipo BLOB.

| Propriedades | Descrição |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |

**TDBRichEdit** - usado para editar/visualizar campos com textos longos e usa formatação de caractere e de paragrafos. Geralmentes são armazenados nos SGBD em campos do tipo BLOB.

| Propriedades | Descrição |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |

**TDBImage** - usado para editar/visualizar imagens. Geralmentes são armazenados nos SGBD em campos do tipo BLOB.

| Propriedades | Descrição |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |

**TDBListBox e TDBComboBox** - conectados a um campo, permitem associar uma lista de strings para preencher o campo. Use quando existe uma lista de valores determinados que podem ser atribuidos ao campo. Exemplo: Se o campo armazena o Estado (UF) de um endereço, poderíamos por todos os estados do Brasil em uma lista onde o usuário escolheria o estado desejado. Usado em campos que podem receber apenas dois valores, como campos lógicos.

| Propriedades | Descrição |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |
| Items | Lista de valores válidos para o campo |

**TDBLookupListBox, TDBLookupComboBox** - são componentes poderosos para tratar de referências a outros arquivos. Visualmente é igual ao *DBComboBox*, a diferença é que a lista de strings é fornecida por outro arquivo. Exemplo: poderíamos usa-los para que o usuário pudesse escolher o código do clientes em uma lista. Essa lista vem do arquivo de clientes, de maneira que podemos ficar seguros que todos os códigos na lista existem no arquivo de clientes. Só que isso não é tudo, podemos ir muito além. Que tal se o que o usuário visse na lista fosse o nome e não o código do cliente? Também é possível, e querendo podemos fazer aparecer na lista o nome e o código do cliente. O componente sabe que na hora de gravar no arquivo é o código e não o nome do cliente que precisa ser gravado.

Preenche um campo com dados contidos em outra tabela. Se o campo mostrado nesses componentes for um campo Lookup, você não precisa especificar nenhuma das propriedades abaixo, apenas DataSource e DataField.

| Propriedades | Descrição |
|---|---|
| DataField | Campo ao qual o controle está ligado |
| DataSource | Componente DataSource que o controle está ligado |
| ListSource | Componente DataSource que contém os valores a serem exibidos na lista |
| ListField | Campo de ListSource que será exibido |
| KeyField | Campo de ListSource usado no relacionamento |

**TDBChart** - componente que exibe valores no formato gráfico.

### TChartSeries

| Propriedades | Descrição |
|---|---|
| DataSource | Componente DataSource que o controle está ligado |
| ColorSource | Campo que contém o valor nunérico para pintar o gráfico |
| XLabelsSource | Campo que será visualizado no titulo da série do gráfico |

# 6 RELATÓRIOS

## *6.1 Impressão de Texto*

A impressão de um relatório contendo somente texto pode ser feita utilizando o método tradicional do Pascal, com poucas modificações:

```
var
  <relatório>: TextFile;
begin
  AssignPrn(<relatório>);
  Rewrite(<relatório>);
  Printer.Title := <título do relatório>;
  Writeln(<relatório>, ...);
          .
          .
          .
  CloseFile(<relatório>);
end;
```

Para cancelar a impressão durante a geração do relatório pode-se utilizar o método Abort do objeto Printer. O objeto Printer e o procedimento AssignPrn estão definidos na unit Printers.

## *6.2 Impressão Gráfica*

A impressão gráfica pura pode ser feita utilizando-se o objeto Printer:

| **Propriedades** | **Descrição** |
|---|---|
| Printing | Informa se existe algum documento sendo impresso pelo processo. |
| Title | Título do relatório |
| PageHeight | Altura da página |
| PageWidth | Largura da página |
| Orientation | Orientação (Landscape ou Portrait) |
| Canvas | É a página propriamente dita |
| PageNumber | Número da página sendo impressa |
| **Métodos** | **Descrição** |
| BeginDoc | Inicia a impressão |
| EndDoc | Finaliza a impressão |
| NewPage | Inicia uma nova página |
| Abort | Cancela uma impressão |

A impressão propriamente dita de cada página é feita no objeto Canvas:

| **Propriedades** | **Descrição** |
|---|---|
| Font | Tipo de letra que será utilizado na escrita |
| Brush | Estilo e cor do preenchimento |
| Pen | Estilo e espessura da linha |
| **Métodos** | **Descrição** |
| TextOut | Escreve um texto em uma determinada posição |
| MoveTo | Move o posição de desenho para uma determinada posição |

| | |
|---|---|
| LineTo | Desenha uma linha da posição atual até a posição determinada |
| PolyLine | Desenha uma série de linhas |
| Arc | Desenha um arco |
| Pie | Desenha um setor de um gráfico tipo torta |
| Rectangle | Desenha um retângulo |
| RoundRect | Desenha um retângulo com bordas arredondadas |
| FillRect | Desenha um retângulo preenchido |
| FloodFill | Preenche uma área qualquer |
| Ellipse | Desenha uma elipse |
| Draw | Coloca um Bitmap, Ícone ou Metafile no Canvas |
| StretchDraw | Coloca um Bitmap, Ícone ou Metafile no Canvas, alterando o tamanho da figura |
| CopyRect | Copia uma área do canvas para outro canvas |

## *6.3 Impressão com o ReportSmith*

O ReportSmith é uma ferramenta que gera relatórios para diversas linguaguens de programação. A ferramenta permite criar relatório a partir de um ambiente gráfico. Cada relatório criado é salvo em um arquivo, que deve acompanhar a aplicação desenvolvida.

Ele possui alguns relatórios padrões, tais como relatórios colunados (mostra os dados em formas de colunas), referência cruzada entre tabelas (mostra os dados tipos uma planilha, associando duas ou mais tabelas), etiquetas (gera etiquetas em formatos padrão), relatórios em modo de página (mostra os registros de uma determinada tabela um por página).

A ferramenta QuickReport é mas disponível gratuitamente com o *Delphi 2.0, mas nas demais* versões do delphi ele não é gratuito.

Através do ReportSmith, e possível gerar relatórios com as seguintes características:

- Combinação de dados entre várias tabelas;
- Ordem e grupo livre de dados através de determinados campos;
- Inserção de cabeçalhos e rodapés;
- Funções de soma, média, máximo e mínimo;
- Criação de caixas de diálogo para inserção de informações, filtragem dos dados;
- Criação de relatórios do tipo master/details;
- Execução de macros durante o relatório; e
- Acesso as mais variadas bases de dados, entre elas:

## *6.4 Impressão com o QuickReport*

A versão 2.0 do Delphi traz uma solução visual para a elaboração rápida de relatórios, o QuickReport. A solução utiliza o editor de formulários do próprio Delphi e se basea no fato de que a maior parte dos relatórios é composta por faixa de informações.

| **Componentes** | **Descrição** |
|---|---|
| QuickReport | É o controlador do relatório.<br>Transforma o formulário do Delphi em um relatório. É ativado pelos métodos *Print* e *Preview*. Permite visualizar o relatório em tempo de desenvolvimento ao se selecionar a opção Preview no menu sensível ao contexto. |
| QRBand | Define as faixas de informações do relatório, que podem ser |
| QRLabel | Define rótulos a serem impressos |
| QRDBText | Imprime campos de um DataSet |
| QRDBCalc | Exibe o somatório, média ou quantidade de registros, valor máximo e mínimo em um campo |
| QRSysData | Imprime informações do sistema como data, hora, número da página, etc. |
| QRShape | Imprime linhas e figuras |
| QRMemo | Imprime um rótulo com mais de uma linha |
| QRDetailLink | Utilizado na criação de relatórios mestre-detalhe |
| QRGroup | Define um grupo de informações |

| | |
|---|---|
| QRPreview | Cria uma área de exibição de relatório, utilizada para preview |
| QRPrinter | É derivado de TPrinter e permite visualizar, salvar e carregar um relatório |

## QuickReport

| **Propriedades** | **Descrição** |
|---|---|
| CollumnMarginInches | Espaço, em 1/10 de polegadas, entre colunas num relatório MultiColuna. |
| CollumnMarginMM | O mesmo que CollumnMarginInches, sendo que em milímetros. |
| Columns | Número de colunas a serem impressas. |
| DisplayPrintDialog | Exibe a caixa de diálogo de impressão quando o método QuickReport.Print é executado. |
| LeftMarginInches | Margem esquerda, em 1/10 de polegadas. |
| LeftMarginMM | O mesmo que LeftMarginInches, sendo que em milímetros |
| Orientation | Orientação do Papel (poPortait ou poLandscape) |
| RestartData | Indica se o QuickReport irá para o primeiro registro do dataset ou inicia a partir do registro corrente. |
| ShowProgress | Indica se o formulário de progresso irá aparecer na criação do formulário. O formulário de progresso possui um botão Cancel. |
| **Métodos** | **Descrição** |
| Preview | Exibe um preview do formulário |
| Print | Imprime o formulário |

## QRBand

| **Propriedade** | **Descrição** |
|---|---|
| BandType | |
| rbTitle | Impresso uma vez no início do relatório |
| rbPageHeader | Impresso uma vez no início de cada página |
| rbPageFooter | Impresso no rodapé de cada página |
| rbColumnHeader | Semelhante ao rbPageHeader exceto pelo fato que é impresso para cada coluna no caso de um relatório multicoluna |
| rbDetail | Replicado para cada registro do DataSet |
| rbSubDetail | Usado como rbDetail em relatórios mestre-detalhe |
| rbGroupHeader | Impresso como cabeçalho de grupo de informações. Deve ser indicado em um QRDetailLink |
| rbGroupFooter | Impresso como rodapé de grupo de informações. Deve ser indicado em um QRDetailLink |
| rbSummary | Impresso no fim do relatório |
| Color | Cor de fundo da faixa |
| Enabled | Indica se o QRBand será impresso |
| Font | Fonte para os controles desse QRBand |
| ForceNewPage | Força que a impressão seja feita numa nova página |
| Frame | Coloca molduras no QRBand |
| LinkBand | Liga QRBand's entre si, forçando que sejam impressos numa mesma página |
| Ruler | Coloca uma régua, em tempo de projeto, no QRBand |
| **Eventos** | **Descrição** |
| BeforePrint | Quando o QRBand vai ser impresso. A impressão pode ser cancelada atribuindo-se False ao parâmetro PrintBand |
| AfterPrint | Ocorre após a impressão ou tentativa de impressão de um QRBand. Se a impressão foi cancelada no evento BeforePrint o parâmetro BandPrinted tem o valor True |

## QRLabel

| **Propriedade** | **Descrição** |
|---|---|
| Alignment | Alinhamento do texto |
| AutoSize | Controle automático do tamanha do componente |
| Caption | Texto a ser impresso |
| Color | Cor de fundo |
| Font | Fonte utilizada |

| | |
|---|---|
| Transparente | Define se o componente será transparent |
| **Eventos** | **Descrição** |
| OnPrint | Ocorre antes do componente ser impresso. O valor a ser impresso esta no parâmetro Value |

## QRDBText

| **Propriedade** | **Descrição** |
|---|---|
| Alignment | Alinhamento do texto |
| AutoSize | Define se o tamanho do componente será ajustado automaticamente de acordo com o seu conteúdo |
| Color | Cor de fundo para o componente |
| DataSource | Fonte de Dados |
| DataField | Campo |
| Font | Fonte |
| Transparent | Indica se o componente será transparente |
| **Eventos** | **Descrição** |
| OnPrint | Ocorre antes do componente ser impresso. O valor a ser impresso esta no parâmetro Value |

## QRDBCalc

| **Propriedade** | **Descrição** |
|---|---|
| AsInteger | Retorna o Valor corrente com inteiro |
| AsReal | Retorna o Valor corrente como real |
| Operation | Operação de Calculo a ser realizada<br>qrcSum - Somatório de um campo<br>qrcCount - Número de registro de um campo<br>qrcMax - Valor máximo de um campo<br>qrcMin - Valor mínimo de um campo<br>qrcAverange - Média de um campo |
| PrintMask | Máscara de saída do QRDBCalc |
| ResetBand | Zera os cálculos sempre for impresso |
| **Eventos** | **Descrição** |
| OnPrint | Antes do componente ser impresso |

## QRSysData

| **Propriedade** | **Descrição** |
|---|---|
| Data | Informação a ser exibida (Hora, Data, Número da Página, Título do Relatório, Contado de Detalhes, Registro Corrente) |
| Alignment | Alinhamento do texto |
| AutoSize | Controle automático do tamanha do componente |
| Color | Cor de fundo |
| Font | Fonte utilizada |
| Transparente | Define se o componente será transparent |
| **Eventos** | **Descrição** |
| OnPrint | Ocorre antes do componente ser impresso. O valor a ser impresso esta no parâmetro Value |

## QRShape

| **Propriedade** | **Descrição** |
|---|---|
| Brush | Cor e Traçado da Figura |
| Pen | Caneta usada na pintura |
| Shape | Tipo da Figura |
| Height | Altura da figura |
| Width | Largura da figura |

| Eventos | Descrição |
|---|---|
| OnPrint | Ocorre antes do componente ser impresso. O valor a ser impresso esta no parâmetro Value |

### QRMemo

| Propriedade | Descrição |
|---|---|
| Alignment | Alinhamento do texto |
| AutoSize | Controle automático do tamanha do componente |
| Lines | Texto a ser impresso |
| WordWrap | Quebra automática de linha de acordo com o tamanho do componente |
| Color | Cor de fundo |
| Font | Fonte utilizada |
| Transparente | Define se o componente será transparent |

### QRDetailLink

| Propriedade | Descrição |
|---|---|
| DataSource | Fonte de Dados na qual um relacionamento mestre-detalhe deve ter sido defenido |
| DeatilBand | QRBand detalhe |
| FooterBand | QRBand rodapé |
| HeaderBand | QRBand cabeçalho |
| Master | QuickReport ou outro QRDetailLink. Um relacionamento mestre-detalhe deve ser definido entre as tabelas envolvidas |
| PrintBefore | Inidica se a informação mestre será impresso antes ou depois dos detalhes |
| **Eventos** | **Descrição** |
| OnFilter | Ocorre a cada movimento do ponteiro de registros no dataset e o parâmetro PrintRecord pode ser utilizado para evitar a impressão do registro corrente |

### QRGroup

| Propriedade | Descrição |
|---|---|
| DataSource | Fonte de Dados |
| DataField | Campo que servirá como base para o grupo |
| FooterBand | Rodapé do grupo |
| HeaderBand | Cabeçalho do grupo |
| Level | Nível do grupo |

### QRPreview

| Propriedade | Descrição |
|---|---|
| PageNumber | Escolha da página a ser exibida |
| Zoom | Escolha do zoom |
| **Métodos** | **Descrição** |
| ZoomToFit | Ajusta o zoom para caber todo o relatório no formulário |
| ZoomToWith | Ajusta o zoom horizontal para caber todo o relatório no formulário |

## 7 CONCLUSÃO

Este trabalho apresentou as características gerais do ambiente de programação Delphi da Borland, sem abordar os fundamentos de lógica de programação e outras peculiaridades de programação visual. Foi enfatizada a forma de se utilizar Delphi juntamente com os recursos de Bases de Dados que podem ser então incluídos nos aplicativos.

Não foi pretendido com essas notas didáticas explorar todos recursos de programação que podem ser obtidos no ambiente Delphi. No entanto, tem-se um conjunto básico das referências de

mecanismos e recursos que podem ser utilizados como suporte nas práticas de aplicativos que precisam de suporte de Bases de Dados.

# Remissivo