Desenvolvendo Sistemas de Informação com Delphi/Kylix

Apostila elaborada por
Eduardo Barbosa de Souza
Equipe de Desenvolvimento
OrionTec Sistemas
ebsouza@oriontec.com.br
www.oriontec.com.br

Viçosa - MG
Setembro de 2003

ÍNDICE

Página

OBSERVAÇÕES

Esta apostila foi elaborada para servir de material de consulta para os alunos. Ela não trás muitos exemplos de aplicativos, pois estes serão desenvolvidos no decorrer do curso.

A apostila está bem resumida, contendo os principais conceitos sobre o Delphi e Kylix. A parte inicial, que trata da linguagem Pascal, deve ser lida principalmente pelos iniciantes, pois é essencial um conhecimento básico desta linguagem para utilizar o Delphi e/ou Kylix.

A parte que trata da VCL (Visual Component Library) e CLX (Component Library for Cross-Platform) contém alguns conceitos importantes e lista as principais propriedades, métodos e eventos, comuns em muitos componentes do Delphi/Kylix. No decorrer do curso serão desenvolvidos programas de exemplo para demonstrar a utilização de muitos componentes.

Os aplicativos de bancos de dados são certamente os mais interessantes na programação em Delphi. Nesta apostila estão alguns conceitos importantes e explicações sobre como é feito o acesso aos dados. Serão desenvolvidos programas de exemplo para demonstrar a criação e o uso de bancos de dados no Delphi e Kylix. Entre estes programas destacam-se:

- **Programa de Cadastro de Clientes:** Será demonstrado como criar um banco de dados e como acessa-lo através do Delphi. Serão utilizadas algumas funções para editar, pesquisar e filtrar os dados.
- **Programa para Controle de Estoque:** Demonstrará alguns recursos avançados do Delphi, para consulta, filtragem e ordenação de dados. Como trabalhar com múltiplas tabelas, mantendo relações entre os campos e como efetuar cálculos com os campos das tabelas.

Serão discutidos conceitos de bancos de dados Cliente/Servidor, e a forma como estes tipos de sistemas são implementados no Delphi/Kylix.

Alguns programas serão desenvolvidos no curso, demonstrando como incluir capacidades de impressão em aplicativos Delphi, inclusive como imprimir relatórios de dados contidos em um banco de dados relacional.

A parte final desta apostila trata da SQL (Structured Query Language). Definindo conceitos e demonstrando alguns exemplos. No decorrer do curso serão utilizadas algumas funções SQL para o trabalho com bancos de dados.

1. CONHECENDO O AMBIENTE DELPHI/KYLIX

1.1. Menus e comandos

Há basicamente três maneiras de acionar um comando no ambiente de desenvolvimento:

· Usar o menu
· Usar a SpeedBar (ou barra de ferramentas).
· Usar um SpeedMenu (um dos menus locais ativados quando se pressiona o botão direito do mouse)

1.1.1. O Menu File

Este menu é um tanto complexo, porque contém comandos que operam em projetos e comandos que operam em arquivos.

Os comandos relacionados aos projetos são New, New Application, Open, Reopen, Save Project As, Save All, Close All. Os comandos relacionados com arquivos de código-fonte são New, New Form, New Data Module, Open, Reopen, Save, Save As, Close e Print. A maioria destes comandos é intuitiva, mas alguns requerem alguma explanação. File > New é um comando geral, que pode ser utilizada para invocar os Experts; iniciar novas aplicações; herdar de formulários existentes; etc.

1.1.2. O Menu Edit

O menu Edit tem algumas operações típicas, tais como as características Undo e Redo, e os comandos Cut Copy e Paste, mais alguns comandos específicos para janelas de formulários ou de editores. O importante a ser notado é que as características padrões do menu Edit trabalham tanto com texto como com componentes do formulário.

Obviamente, você pode copiar e colar texto no editor ou pode copiar e colar componentes num formulário, ou de um formulário para outro. Você pode até mesmo copiar e colar para uma janela de origem diferente do mesmo formulário, como um painel ou uma caixa de grupo.

Juntamente com os comandos típicos do menu Edit, o Delphi inclui uma série de outros comandos, os quais, em sua maioria, relacionam-se a formulários. As operações específicas para os formulários também podem ser acessadas por meio do SpeedMenu (ou menu local) de formulários. Um comando não replicado no menu local do formulário é Lock Controls, que é muito útil para evitar uma alteração acidental na posição de um componente no formulário. De fato, você pode tentar dar um clique duplo em um componente e acabar movendo-o. Uma vez que não existe operação de Undo em formulários, proteger-se de erros similares bloqueando os controles pode ser muito útil

1.1.3. O Menu Search

O menu Search também tem alguns comandos padrões, tais como Search e Replace. Outros comandos não são tão simples de entender. O comando Incremental Search é um deles. Quando você seleciona este comando, em vez de mostrar uma caixa de diálogo em que pode introduzir o texto que quer encontrar, o Delphi muda para o editor. Lá, você pode digitar o texto que quer procurar diretamente na área de mensagem do editor.

Quando você digitar a primeira letra, o editor se moverá para a primeira palavra que se inicia com essa letra (Mas se o seu texto procurado não for encontrado, as letras que digitou nem mesmo serão exibidas na área de mensagem do editor). Se esta não for a palavra que você está procurando, continue digitando; o cursor continuará a pular para a palavra seguinte que se inicia com essas letras.

Os comandos Go to Line Number e Show Last Compile Error são muito intuitivos. O comando Find Error pode parecer estranho a princípio. Ele não é usado para procurar um erro do compilador, mas para encontrar um erro em tempo de execução. Quando você está executando um programa independente e chega a um erro muito sério, o Delphi exibe um número de endereço interno (ou seja, o endereço lógico do código compilado). Você pode introduzir este valor na caixa de diálogo Find Error para fazer com que o Delphi compile novamente o programa procurando o endereço específico. Quando encontra o endereço, o Delphi mostra a linha do código-fonte correspondente.

O último comando do menu Search, Browse Symbol, invoca o Browser, uma ferramenta que você pode usar para explorar um programa compilado, analisando muitos detalhes Para entender a saída do Browser, você precisa entender em profundidade a linguagem Object Pascal e a Visual Components Library (VCL).

1.1.4. O Menu View

A maioria dos comandos View pode ser usada para exibir uma das janelas do ambiente Delphi, como por exemplo o Project Manager, a lista de Breakpoints ou a lista de Components. Estas janelas não se relacionam. Algumas são usadas durante a depuração; outras, quando você está escrevendo o código.

Os comandos seguintes do menu View são importantes, razão pela qual os comandos Toggle, Units e Forms também estão disponíveis na SpeedBar padrão. Toggle é usado para alternar entre o formulário em que você está trabalhando e seu código-fonte.

O comando da janela New edit duplica as janelas de edição e seu conteúdo. É a única forma de visualizar dois arquivos lado a lado no Delphi, já que o Delphi usa abas para mostrar os múltiplos arquivos que você pode carregar. Uma vez que você tenha duplicado a janela de edição, cada uma pode conter diferentes arquivos.

Os dois últimos comandos do menu View podem ser usados para ocultar a SpeedBar ou a paleta Components.

1.1.5. O Menu Project

O próximo menu suspenso, Project possui comandos para manipular um projeto e compilá-lo. Add to Project e Remove from Project são dois comandos intuitivos utilizados para adicionar e remover formulários ou código-fonte Pascal em um programa.

O comando Compile constrói ou atualiza o arquivo executável da aplicação, verificando quais arquivos-fonte foram alterados. Com Build All, você pode solicitar ao Delphi que compile cada arquivo de código-fonte do projeto, mesmo que não tenham sido alterados desde a última compilação. Se você somente quiser saber se a sintaxe do código que você escreveu está correta, mas não quiser construir o programa você pode utilizar o comando Syntax Check. O comando, Information, exibe alguns detalhes sobre a última compilação feita.

No final, vem o menu Options, utilizado para definir algumas opções de projeto, tais como opções de compilador e linker, opções do objeto Application e assim por diante. Quando você define opções de projeto, pode marcar a caixa Default para indicar que o mesmo conjunto de opções será utilizado para seus próximos novos projetos.

1.1.6. O Menu Run

O menu Run também poderia ser chamado Debug. A maioria dos comandos está relacionada à depuração, inclusive o próprio comando Run.

Quando você executa um programa dentro do ambiente Delphi, executa-o sob o depurador integrado (a menos que tenha desabilitado a opção Environment correspondente). O comando Run, e o correspondente ícone SpeedBar, está entre os comandos mais comumente usados, uma vez que o Delphi recompila automaticamente um programa antes de executá-lo - pelo menos se o código-fonte tiver mudado.

O comando seguinte, Parameters, pode ser usado para passar alguns parâmetros da linha de comando para o programa que você vai executar. Os comandos restantes são todos usados durante a depuração; deste modo, execute o programa passo a passo, estabeleça pontos de interrupção, inspecione os valores das variáveis e dos objetos e assim por diante. Alguns desses comandos de compilação estão disponíveis diretamente no SpeedMenu do editor.

1.1.7. O Menu Component

Os comandos do menu Component podem ser utilizados para escrever componentes, adicioná-los à biblioteca VCL, ou configurar a biblioteca ou a Component Palette. O comando New invoca o simples Component Expert.

1.1.8. O Menu Database

O menu Database reúne as ferramentas do Delphi relacionadas com bancos de dados, desde os internos (tal como o Form Wizard) até programas externos separados, tal como o Database Explorer. Os comandos neste menu dependem de sua edição do Delphi A edição Enterprise, é claro, possui mais entradas neste menu do que outras edições

1.1.9. O Menu Tools

O menu Tools deve ser utilizado para configurar opções do ambiente de desenvolvimento e do editor de código-fonte, além de permitir executar uma série de programas externos e ferramentas. Você pode utilizar o comando Tools para configurar e adicionar novas ferramentas ao menu suspenso.

1.1.10. Solicitando Ajuda

Há basicamente duas maneiras de invocar o sistema Help: selecione o comando adequado no menu suspenso Help, ou escolha um elemento da interface Delphi ou um símbolo no código-fonte e pressione a tecla Fl.

Note que não há apenas um único arquivo Help no Delphi. Na maioria das vezes, você invocará o Help do Delphi, mas este arquivo é complementado pelo arquivo de Help do Object Pascal, MS Windows API Help e pelo Help do Component Writer.

Você pode encontrar quase tudo no sistema Help, mas precisa saber o que procurar. Juntamente com os três arquivos Help principais, há uma série de outros arquivos relacionados a tópicos específicos, tais como o Creating Windows Help, ou a ferramentas, tais como o WinSight Help.

Os arquivos Help oferecem uma série de informações tanto para programadores iniciantes como para especialistas, mas elas são especialmente valiosas como ferramenta de referência. Elas listam todos os métodos e propriedades de cada componente, os parâmetros de cada método ou função, e detalhes semelhantes, os quais são particularmente importantes quando se está escrevendo código.

1.2. A SpeedBar

Alguns comandos freqüentemente usados estão disponíveis na SpeedBar (Barras de Ferramentas). Se os comandos que você usa muito não estiverem lá, é hora de personalizar a SpeedBar a fim de que ela realmente o ajude a usar o Delphi mais eficientemente.

Você pode mudar facilmente o tamanho da SpeedBar ao arrastar a linha espessa entre ela e a paleta Components. As outras operações que você pode efetuar com a SpeedBar (além de removê-la) são adicionar, remover ou substituir os ícones ao usar o comando Properties do SpeedMenu da própria SpeedBar. Esta operação invoca o SpeedBar Editor.

Para adicionar um ícone à SpeedBar, você precisa simplesmente encontrá-lo sob a categoria (correspondente a um menu suspenso) e arrastá-lo para a barra. Da mesma maneira, você pode arrastar um ícone para fora da SpeedBar ou simplesmente mudá-lo para outro local.

1.3. Os SpeedMenus

O Delphi tem um grande número de itens de menu, porém nem todos os comandos estão disponíveis através dos menus suspensos. Às vezes, você precisa usar os SpeedMenus (menus locais) para janelas ou áreas de janelas específicas. Para ativar o SpeedMenu, pressione o botão direito do mouse sobre o elemento da interface com o usuário ou pressione as teclas Alt e F10.

Quase todas as janelas no Delphi (com a exclusão das caixas de diálogo) possui seu próprio SpeedMenu com comandos relacionados.

1.4. Trabalhando com Formulários

O desenho de formulários é a parte central do desenvolvimento visual no ambiente Delphi. Todo componente que você coloca num formulário ou qualquer propriedade que estabelece é armazenada num arquivo que descreve o formulário (um arquivo DFM) e também exerce algum efeito sobre o código-fonte associado ao formulário (o arquivo PAS).

Quando você inicia um projeto novo, em branco, o Delphi cria um formulário vazio, e você pode começar a trabalhar com ele. Você também pode começar com um formulário existente (usando os vários modelos disponíveis) ou adicionar novos formulários a um projeto.

Quando você está trabalhando com um formulário, pode operar sobre suas propriedades, sobre um de seus componentes ou sobre diversos componentes ao mesmo tempo. Para selecionar o formulário ou um componente, você pode simplesmente dar um clique sobre ele ou usar o Object Selector (a caixa de opções no Object Inspector), onde você sempre pode ver o nome e o tipo do item selecionado você pode selecionar mais de um componente pressionando e mantendo pressionada a tecla Shift enquanto seleciona os componentes com o botão esquerdo do mouse, ou arrastando um retângulo de seleção no formulário.

Assim que você estiver trabalhando em um formulário, o SpeedMenu terá uma série de características úteis. Você pode utilizar os comandos Bring to Front e Send to Back para mudar a posição relativa de componentes do mesmo tipo. Quando tiver selecionado mais de um componente, você poderá alinhá-los ou reajustá-los. A maioria das opções na caixa de diálogo Alignment também está disponível na Alignment Palette.

A partir do SpeedMenu, você pode abrir duas caixas de diálogo para fixar a ordem de tabulação dos controles visuais e a ordem de criação dos controles não-visuais. Você pode utilizar o comando Add to Repository para adicionar o formulário em que estiver trabalhando em uma lista de formulários disponíveis para utilização em outros projetos. Finalmente, pode utilizar o comando View as Text para fechar o formulário e abrir sua descrição textual no editor. Um comando correspondente no SpeedMenu do editor reverterá a situação.

Juntamente com os comandos SpeedMenu específicos, você pode fixar algumas opções de formulário ao usar o comando Tools > Options e ao escolher a página Preferences. As opções relacionadas aos formulários estão listadas sob o título Form Designer (ativação de grade e tamanho). Quando você ativa a grade, pode mover componentes do formulário somente em posições fixas (separadas pelo número de pixeis do tamanho da grade) e fixar o tamanho delas aumenta a intervalos fixos Sem uma grade, raramente você será capaz de alinhar dois componentes manualmente (usando o mouse).

Há duas alternativas ao uso do mouse para fixar a posição de um componente:

· Fixe o valor para as propriedades Left e Top.

· Use as teclas de direção enquanto mantém pressionada a tecla Ctrl .

Usar o método Ctrl-tecla de direção é particularmente útil para fazer a sintonia fina da posição de um elemento. (A opção Snap to Grid funciona somente para operações com o mouse.) Similarmente, ao pressionar as teclas de direção enquanto mantém pressionada a tecla Shift, você pode fazer a sintonia fina do tamanho de um componente.

Tabela 1.1: Os comandos de SpeedMenu adicionados quando alguns componentes do Delphi estão selecionados:

Comando de menu	Componentes
Menu Designer	MainMenu, PopupMenu
Fields Editor	Table, Query, StoredProc
Database Editor	Database
Execute	BatchMove
Columms Editor	DBCrid
ImageList Editor	ImageList
New Page	PageControl
Next Page	PageControl, NoteBock, TabbedNoteBook
Previous Page	PageControl, NoteBock, TabbedNoteBook
Properties	Qualquer controle OCX

1.5. A Paleta Components

Quando quiser adicionar um novo componente a um formulário em que está trabalhando, você pode dar um clique sobre um componente numa das páginas da paleta Components, depois dar um clique sobre o formulário para incluir o novo componente. No formulário, você pode pressionar o botão esquerdo do mouse e arrastar o mouse para deixar a posição e o tamanho do componente de uma vez por todas, ou apenas dar um clique, para permitir que o Delphi use um tamanho padrão.

Cada página da paleta tem uma série de componentes, indicados por um Ícone, e um nome, o qual aparece como uma dica Instantânea. Esses são os nomes oficiais dos componentes. Na realidade, são os nomes das classes que definem o componente, sem a inicial T (por exemplo, a classe é TButton, o nome é Button).

1.6. O Object Inspector

Quando estiver projetando um formulário, para estabelecer a propriedade de um componente (ou o próprio formulário), use o Object Inspector. Sua janela lista as propriedades do elemento selecionado e seus valores em duas colunas cujo tamanho pode ser mudado. Um Object Selector na parte superior do Object Inspector indica o componente atual e seu tipo de dados, e este seletor pode ser usado para mudar a seleção atual.

É importante notar que o Object Inspector não lista todas as propriedades de um componente. Ele inclui somente as propriedades que podem ser definidas em tempo de projeto. Outras propriedades são acessíveis somente em tempo de execução. Para conhecer todas as diferentes propriedades de um componente, consulte os arquivos Help.

Uma característica interessante do Object Inspector é que a coluna direita permite a correta edição do tipo de dados da propriedade. Dependendo da propriedade, você será capaz de selecionar uma String, inserir um número, introduzir um valor ou escolher de uma lista de opções (indicado por uma seta semelhante a uma caixa combinada), ou invocar um editor específico (indicado por um botão com reticências).

Algumas propriedades tais como Fonte, podem ser personalizadas tanto expandindo suas subpropriedades (indicada por um sinal de mais ou menos à esquerda do nome) ou invocando um editor. Quando uma opção é particularmente complexa, o Object inspector tem editores especiais que você pode ativar pressionando o pequeno botão com três pontos.

Em alguns casos, como em listas de strings, os editores especiais são a única maneira de selecionar uma propriedade. O mecanismo de subpropriedade está disponível em dois casos diferentes: com conjuntos e classes. Quando você expande propriedades, cada uma delas possui seu próprio comportamento no Object inspector, novamente dependendo de seu tipo de dado.

Você usará o Object Inspector freqüentemente. Ele deve estar sempre visível quando você editar um formulário, mas também pode ser útil olhar para os nomes dos componentes e propriedades enquanto você está escrevendo o código.

1.7. A Paleta Alignment

A última ferramenta relacionada ao desenho do formulário é a paleta Alignment. Você pode abrir esta paleta com o comando Alignment Palette do menu View. Como uma alternativa, você pode escolher os componentes que quer alinhar e depois acionar o comando Align do SpeedMenu do formulário.

A paleta Alignment apresenta uma série de comandos para posicionar os vários controles, centralizá-los, espaçá-los igualmente e assim por diante. Para ver o efeito de cada botão, simplesmente mova o mouse sobre a janela e olhe para as dicas instantâneas.

1.8. Escrevendo Código

Quando tiver desenhado um formulário no Delphi, usualmente você precisará escrever algum código para responder a alguns de seus eventos. Todas as vezes que você trabalhar num evento, o Delphi abrirá o editor com o arquivo-fonte relacionado ao formulário.

O editor do Delphi permite-Ihe trabalhar em diversos arquivos de código-fonte de uma só vez, usando uma metáfora de "fichário com abas". Cada página do fichário corresponde a um arquivo diferente. Você pode trabalhar em units relacionadas aos formulários, units independentes de código Pascal, arquivos do projeto; pode abrir os arquivos de descrição do formulário em formato textual e até mesmo trabalhar com arquivos de texto puros.

Há uma série de opções disponíveis para o ambiente do editor, a maioria localizada nas páginas Editor Options, Editor Display e Editor Colors da caixa de diálogo Environment Options. Na página Preferences, você pode estabelecer a característica Autosave para os arquivos do editor.

As outras três páginas de opções do editor podem ser usadas para definir as configurações padrões do editor (escolhendo-se entre diferentes funções de mapeamento de toques de teclas), destaque de sintaxe e fonte.

O SpeedMenu da janela do editor possui muitas opções para depuração e outras opções relacionadas com o próprio editor, tais como comandos para fechar a página atual, abrir o arquivo sob o cursor, visualizar ou ocultar o painel de mensagem abaixo da janela e invocar as opções do editor discutidas anteriormente.

1.9. O Gerenciador de Projetos

Quando um projeto é carregado, escolha o comando Project Manager (Gerenciador de Projetos) do menu View para abrir uma janela de projeto. A janela lista todos os formulários e units que compõem o atual projeto. Quando você seleciona um arquivo-fonte, a janela Project Manager exibe o nome dos arquivos correspondentes e suas estampas de hora/data.

O SpeedMenu do Project Manager permite-lhe executar uma série de operações no projeto, como, por exemplo, adicionar arquivos novos ou existentes, remover arquivos, visualizar um arquivo de código-fonte ou um formulário e adicionar o projeto ao repositório. A maioria destes comandos também está disponível na SpeedBar desta janela (não na SpeedBar principal do Delphi).

1.10. Estabelecendo Opções de Projeto

A partir do Project Manager (ou do menu Options), você pode invocar as Project Options (Opções de Projeto). A primeira página das Project Options, denominada Forms, lista os formulários que devem ser criados automaticamente no início do programa (o comportamento padrão) e os formulários que são criados manualmente pelo programa. Você pode facilmente mover um formulário de uma lista para outra.

A página Application é usada para definir o nome da aplicação, o nome de seu arquivo Help e o ícone do programa. Outras escolhas de Project Options relacionam-se ao compilador e ao linkeditor do Delphi

Todas as Project Options são salvas automaticamente com o projeto, mas num arquivo separado com uma extensão DOF. Este é um arquivo de texto que você pode facilmente editar Não apague este arquivo se você tiver mudado qualquer uma das opções padrões.

1.11. Compilando um Projeto

Há diversas maneiras de compilar um projeto. Se você executá-lo (pressionando F9 ou dando um clique sobre o ícone da SpeedBar), o Delphi o compilará primeiro. Quando o Delphi compila um projeto, ele compila somente os arquivos que foram mudados. Se, em vez disso, você selecionar o comando Build All do menu Compile, cada arquivo será compilado, mesmo que não tenha sido modificado. Este segundo comando raramente é usado, uma vez que o Delphi usualmente determina quais arquivos foram mudados e compila-os conforme o necessário.

O projeto lista os arquivos de código-fonte que são parte dele, eventualmente com os formulários relacionados. Esta lista é visível tanto na fonte do projeto como no Project Manager, e é utilizada para compilar ou reconstruir um projeto. Primeiro, cada arquivo de código-fonte é transformado em uma unit compilada do Delphi, um arquivo com o mesmo nome do arquivo-fonte Pascal e a extensão DCU. Por exemplo, UNITI.PAS é compilado em UNITI.DCU

As units compiladas que constituem o projeto são incorporadas (ou vinculadas) ao arquivo executável juntamente com código da biblioteca VCL quando o próprio código-fonte do projeto é compilado. Você poderá entender melhor o processo de compilação e acompanhar o que acontece durante esta operação se ativar a opção Show Compiler Progress. Esta opção se encontra na página Preferences da caixa de diálogo Environment Options, sob o título Compiling. Apesar de tornar a compilação um pouco mais lenta, a janela Compile permite que você veja quais arquivos-fonte são compilados cada vez que o programa é executado.

1.12. Explorando um Programa Compilado

O Delphi oferece uma série de ferramentas que você pode usar para explorar um programa compilado, inclusive o depurador e o Browser.

1.13. Depurador Integrado

O Delphi tem um depurador integrado, o qual possui um número enorme de características. Porém, também é possível comprar um depurador independente poderoso, chamado Turbo Debugger.

Você não precisa fazer muita coisa para usar o depurador integrado. De fato, cada vez que você rodar um programa a partir do Delphi, ele será executado no depurador. Isto significa que você pode estabelecer um ponto de interrupção para interromper o programa quando ele atingir uma linha de código estabelecida.

Quando um programa é interrompido, você pode inspecionar seu status detalhadamente. Você pode analisar o valor de uma variável (com o comando Evaluate), prestar atenção a seu valor (para ver quando ele muda) ou até mesmo verificar as chamadas de funções na pilha.

1.14. O Object Browser

Quando tiver compilado um programa (mesmo que não o esteja executando ou depurando), você pode executar o Object Browser para explorá-lo. Esta ferramenta permite-Ihe ver todas as classes definidas pelo programa (ou pelas units usadas direta e indiretamente pelo programa), todos os nomes e variáveis globais e assim por diante. Para cada classe, o Object Browser mostra a lista de propriedades, métodos e variáveis - tanto privadas como herdadas, privadas ou públicas.

Mas as características do Object Browser não significarão muito até que você entenda as capacidades orientadas a objeto da Linguagem Object Pascal usada pelo Delphi.

1.15. Ferramentas Adicionais do Delphi

O Delphi oferece muito mais ferramentas para seus esforços de programação. Por exemplo, o Menu Designer é uma ferramenta visual usada para criar a estrutura de um menu. Também há vários Experts, usados para gerar a estrutura de um aplicativo ou de um novo formulário.

Outras ferramentas são aplicativos independentes relacionados ao desenvolvimento de um aplicativo MS Windows, tais como o Image Editor e WinSight, um programa espião que Ihe permite ver o fluxo de mensagem do Windows.

Existem muitas ferramentas de bancos de dados externos, tais como Database Desktop e o Database Explorer. Um programador pode utilizar outras ferramentas de terceiros para cobrir áreas frágeis do Delphi. Por exemplo, IBExpert para manipular bancos de dados do Interbase.

1.16. Arquivos produzidos pelo sistema

O Delphi produz uma série de arquivos para você, e é preciso ter ciência de como ele nomeia esses arquivos. Há basicamente dois elementos que exercem impacto sobre como os arquivos são nomeados:

· Os nomes que você dá a um projeto e seus formulários.

· As extensões predefinidas usadas pelo Delphi para os arquivos que você escreve e para aqueles gerados pelo sistema.

Tabela 1.2: As extensões dos arquivos Delphi

Extensão	Tipo de Arquivo	Criação	Necessidade de Recompilação	Descrição
BMP, ICO	Arquivos gráficos	Desenvolvimento (Image Editor)	Não, mas podem ser exigidos em tempo de execução	Bitmaps e ícones usualmente são incluídos no código, mas você não os apagará. Eles são usados para glifos de botões ou caixas de listagem, componentes de imagens e assim por diante
DCU	Unidade compilada Delphi	Compilação	Não, mas se a fonte não tiver sido mudada, eles agilizam um processo de reconstrução	Estes arquivos de objetos são o resultado da compilação de um arquivo PAS (e o formulário correspondente)
~DF	Backup de fomulário gráfico	Desenvolvimento	Não	Este é o arquivo de backup do arquivo DFM do formulário
DFM	Arquivo de formulário gráfico Delphi	Desenvolvimento	Sim	Este é o arquivo binário com a descrição das propriedades de um formulário e dos controles que ele mantém. O arquivo é convertido para um formato textual automaticamente quando você o carrega no editor Delphi.
DOF	Arquivo de opções do Delphi	Desenvolvimento	Necessário somente se opções especiais foram configuradas	Este é um arquivo com as configurações atuais para as opções do projeto. A extensão era OPT no Delphi 1.
~DP	Backup do projeto	Desenvolvimento	Não	Este arquivo é gerado automaticamente quando uma nova verão de um arquivo de projeto salva. Ele é uma cópia da versão mais antiga do arquivo.
DPR	Arquivo de projeto Delphi	Desenvolvimento	Sim	No Delphi, o arquivo do projeto encontra-se em código-fonte Pascal. Ele lista todos os elementos de um projeto e oferece algum código de inicialização.
DSK	Configurações de desktop	Desenvolvimento	Não	Este arquivo contém informações sobre a posição das janelas Delphi, os arquivos abertos no editor e outras configurações de desktop, inclusive algumas opções de ambiente.
DSM	Dados do	Compilação	Não, usada pelo	Este arquivo armazena todas as

	Object Browser	(somente se a opção tiver sido definida)	Browser	informações do Browser e pode ser usado para acessar essas informações sem recompilar o projeto (ou quando você não puder recompilá-lo por causa de erro).
EXE	Arquivo compilado executável	Compilação	Não	Este é o resultado de seus esforços: o arquivo executável de seu aplicativo Ele inclui todas as units compiladas, recursos e formulários
~PA	Backup da unidade	Desenvolvimento	Não	Este arquivo armazena um backup de uma unidade. Ele é gerado automaticamente pelo Delphi
PAS	Código-fonte da unit	Desenvolvimento	Sim	Este arquivo contém o código-fonte de uma unit Pascal, que pode ser o código-fonte de um formulário, ou um arquivo-fonte independente. Contém a definição da classe para o formulário e o código de seus manipuladores de eventos.
RES	Arquivo de recursos compilado	Desenvolvimento	Sim	O arquivo binário associado ao projeto e que contém, por padrão, seu ícone. Você pode adicionar outros recursos a este arquivo ou outros arquivos deste tipo ao projeto.

A maioria desses arquivos é formada por pequenos arquivos, incluindo os arquivos-fonte de backup, as opções e o arquivo desktop. As units compiladas são ligeiramente maiores.

O arquivo DSM guarda informações do Browser que permitem o uso do Object Browser, mesmo quando você tiver mudado o código-fonte (mas sempre depois de uma compilação bem-sucedida). Isto pode ser particularmente útil quando você acidentalmente introduzir um erro no código, evitando que o compilador construa uma nova versão do programa. Arquivos DSM, entretanto, podem facilmente tornar-se grandes, e você pode evitar que sejam criados marcando a opção Desktop Only em vez de Desktop and Symbols na página Preferences da caixa de diálogo Environment Options. Isto também reduzirá o tempo para compilar/vincular.

A grande vantagem do Delphi sobre outros ambientes de programação visual é que a maioria dos arquivos de código-fonte é composta de arquivos de texto ASCII puros. Os arquivos Desktop usam uma estrutura semelhante à dos arquivos INI do MS Windows, em que cada seção é indicada por um nome escrito entre colchetes.

Além de opções de ambiente e posições de janela, o arquivo desktop também contém uma serie de listas de históricos (listas de arquivos de certo tipo, os quais estão disponíveis nas caixas de diálogo abertas), e uma indicação de pontos de interrupção atuais, visualizações, módulos ativos, módulos fechados e formulários.

2. A LINGUAGEM PASCAL

Quando a linguagem Pascal foi projetada, existiam poucas linguagens de programação (entre elas, FORTRAN, COBOL e BASIC). A idéia-chave da nova linguagem era ordem, administrada por meio de um forte conceito de tipo de dados.

2.1. Tipos de Dados Pascal

A linguagem Pascal original foi baseada em algumas noções simples, as quais se tornaram muito comuns nas linguagens de programação. A primeira é a noção de *tipo de dados.* O tipo determina os valores que uma variável pode ter (e a representação interna desses valores) e as operações que podem ser realizadas com essa variável. Exemplo de declarações de dados:

```
var
  Valor   : Integer;
  Correto : Boolean;
  A, B    : Char;
```

A palavra-chave **`var`** pode ser usada em diversos lugares no código, como, por exemplo, no início do código de uma função. Depois de **`var`** vem uma lista de nomes de variáveis, seguidos de um ponto-e-vírgula (;) e do nome do tipo de dados. Você pode escrever mais de um nome de variável numa única linha, como na última linha do código acima.

Assim que tiver definido uma variável de determinado tipo, você poderá efetuar com ela somente as operações suportadas por seu tipo de dados. Por exemplo, você pode usar o valor booleano (Boolean) num teste e um valor inteiro (Integer) numa expressão numérica. Porém você não pode fazer o inverso.

Usando atribuições simples, podemos escrever o seguinte código:

```
Valor := 10;
Correto := True;
```

Mas a instrução seguinte não está correta, porque as duas variáveis tem diferentes tipos de dados:

```
Valor := True;
```

Se você tentar compilar este código, o Delphi 2 exibirá um erro de compilador com esta descrição: *Incompatible types: 'Integer'* and *'Boolean'*. As mensagens de erro do Delphi 2 são bem detalhadas, e habilitando também dicas e avisos do compilador, você geralmente é capaz de entender melhor os erros no código. Usualmente, erros como este são erros de programação, porque não faz sentido atribuir um valor True ou False a um número inteiro. Você não deve culpar o Delphi por isso. Ele somente o avisa de que há algo errado no código.

Logicamente sempre é possível converter o valor de um tipo num tipo diferente. Em alguns casos esta conversão é automática, mas usualmente você precisa chamar um procedimento especifico do sistema que mude a representação interna dos dados.

2.2. Tipos de Dados Predefinidos

Ha diversos tipos de dados predefinidos, os quais podem ser divididos em três diferentes grupos: tipos ordinais, tipos reais e strings.

2.2.1. Tipos Ordinais

Os tipos ordinais têm a noção de ordem. Isto não somente se relaciona ao fato de que você pode comparar dois valores para ver qual é o mais elevado, mas também que você pode solicitar o valor seguinte ou precedente de determinado valor ou computar o valor mais baixo ou mais elevado.

Os três tipos ordinais predefinidos mais importantes são Integer, Boolean e Char (inteiro, booleano e caractere, nessa ordem). Entretanto, existem diversos outros tipos relacionados que possuem o mesmo significado mas uma representação interna e limites diferentes.

Eis uma lista completa dos tipos ordinais:

· Integer, Cardinal, ShortInt, SmallInt, LongInt, Byte, Word
· Boolean, Bytebool, Wordbool, Longbool
· Char, ANSIChar, WideChar

Os vários tipos Integer correspondem a diferentes representações internas. ShortInt, SmallInt e LongInt representam números com sinais com várias representações; Byte e Word representam um valor sem sinal; e Integer e Cardinal correspondem a um valor baseado na representação nativa (2 bytes em uma plataforma 16 bits, 4 bytes em uma plataforma 32 bits), com e sem sinal, respectivamente.

Os quatro tipos Boolean são necessários para programação em Windows, mas Boolean é mais freqüentemente utilizado

Os três tipos de caractere são utilizados para indicar caracteres de 8 bits (ANSI) ou 16 bits (Unicode), e o tipo default.

2.2.2. O Exemplo Limites

Para dar-lhe uma idéia das diferentes faixas de alguns dos tipos ordinais, escrevi um programa Delphi simples, chamado LIMITES.

O programa LIMITES é baseado em um formulário simples, que possui seis botões (cada um nomeado com um tipo de dado ordinal) e alguns rótulos para as categorias de informação. Alguns dos rótulos são utilizados para armazenar texto estático, outros para exibir a informação sobre o tipo cada vez em que um dos botões é pressionado.

Todas as vezes que você pressionar um dos botões à direita, o programa atualizará os três rótulos. Estes rótulos sustentam o número de bytes da representação do tipo de dados e o valor máximo e o valor mínimo que o tipo de dados pode armazenar. Cada botão tem seu próprio método de resposta a eventos OnClick, mas o código usado para computar os três valores é ligeiramente diferente de botão para botão. Por exemplo, eis o código-fonte do evento OnClick para o botão de número inteiro (IntButton):

```
Procedure TForml.IntButtonClick ( Sender: TObject );
var
   Numero: Integer;
begin
   TipoLabel.caption := 'Integer';
   SizeLabel.Caption := IntToStr (SizeOf (Numero));
   MaxLabel.Caption := IntToStr (High (Numero));
   MinLabel.Caption := IntToStr (Low (Numero));
end;
```

Se você tiver alguma experiência em programação Delphi, poderá examinar o código-fonte do programa para entender como ele funciona. Para os iniciantes, basta notar o uso de três funções: SizeOf, High e Low. Os resultados das duas últimas funções são ordinais do mesmo tipo (neste caso, integers), e o resultado da função SizeOf é sempre um Integer, assim, eles são primeiro traduzidos em strings utilizando a função IntToStr, e então copiados para as legendas dos três botões.

Há algumas rotinas do sistema - rotinas definidas na linguagem Pascal e na unit do sistema Delphi - que trabalham com números ordinais, como se pode ver na Tabela 2.1.

Tabela 2.1: Rotinas do sistema para tipos ordinais

Rotina	Propósito
Dec	Efetua um decremento da variável passada por parâmetro, por um ou por um eventual segundo parâmetro
Inc	Incrementa a variável passada por parâmetro, por um ou pelo valor especificado
Odd	Retoma True se o argumento for um número ímpar
Pred	Retoma o valor antes do argumento na ordem determinada pelo tipo de dados, o predecessor
Succ	Retorna o valor após o argumento, o sucessor
Ord	Retorna um número indicando a ordem do argumento dentro do conjunto de valores do tipo de dados
Low	Retorna o valor mais baixo na faixa do tipo ordinal passado por parâmetro
High	Retorna o valor mais elevado do tipo de dado ordinal

2.2.3. Tipos Reais

Os tipos reais representam números com pontos flutuantes em vários formatos. O menor tamanho de armazenagem é dado pelos números Single. Depois há os números Real, os números Double e os números Extended. Todos eles são tipos de dados de ponto flutuante com diferentes precisões. Há também dois tipos de dados estranhos, o Comp, que descreve números inteiros demasiadamente longos e Currency. Como o nome implica, o tipo de dado Currency (moeda) foi adicionado para exibir valores monetários imensos sem perder o menor dígito significante (um problema comum com valores de ponto flutuante). A representação interna dos tipos de dados comp e currency é similar a tipos inteiros, embora eles não possam ser considerados tipos ordinais pela sua imensa faixa de valores.

Não podemos criar um programa semelhante a LIMITES com tipos de dados reais, porque não podemos usar as funções High e Low ou a função Ord em variáveis do tipo real Os tipos reais representam (teoricamente) um conjunto infinito de números, os números ordinais representam um conjunto fixo de valores. Por esta razão, faz sentido perguntar a ordem do caractere '*n*' no tipo de dados char, mas não faz nenhum sentido perguntar a ordem de 7143.l52 no tipo de dados real.

Embora você possa saber de fato se um número real tem um valor mais elevado do que outro, não faz sentido perguntar quantos números reais existem antes de determinado número (este é o significado da função Ord). A mesma discussão pode ser estendida a outras funções disponíveis somente para tipos ordinais.

Tipos reais são usados quase exclusivamente em programas envolvendo fórmulas matemáticas, e eles usualmente têm um papel limitado na parte de interface com o usuário do código (o lado Windows). O Delphi usa tipos reais no tipo de dados TDateTime. Este é um tipo de ponto flutuante, o qual é o único tipo que tem uma faixa bastante ampla de valores para armazenar dias, meses, anos, horas, minutos e segundos, até uma resolução de milissegundos numa única variável.

2.3. Typecasting e Conversões de Tipos

Você não pode atribuir uma variável a outra de um tipo diferente. No caso de você precisar fazer isso, há duas opções. A primeira opção é o *typecasting*, a qual usa uma notação funcional simples, com o nome do tipo de dados de destino:

```
var
  N: Integer;
  C: Char;
  B: Boolean;
begin
   N := Integer ('x');
   C := Char (N);
   B := Boolean (0);
```

```
end;
```

Geralmente você pode usar o typecasting com tipos de dados que têm o mesmo tamanho. Usualmente é seguro fazer isso entre tipos ordinais ou entre tipos reais, mas você também pode trabalhar entre tipos indicadores (e também objetos), contanto que saiba o que está fazendo. O typecasting, porém, geralmente é uma prática perigosa de programação, porque permite-lhe acessar um valor como se ele representasse algo mais. Uma vez que a representação interna dos tipos de dados e a dos objetos geralmente não combinam, você se arrisca a erros difíceis de ser rastreados. Por esta razão, você deve evitar o typecasting, ou recorrer às técnicas seguras para usá-lo entre objetos oferecidos por métodos do tipo em tempo de execução.

A segunda opção é usar uma rotina de conversão de tipos As rotinas para os vários tipos de conversões estão resumidas na Tabela 2.2.

Tabela 2.2: Rotinas do sistema para conversão de tipos

Rotinas	**Propósito**
Chr	Converte um número ordinal em caractere
Ord	Converte um valor do tipo ordinal num número que indica sua ordem
Round	Converte um valor do tipo real num valor do tipo inteiro, arredondando seu valor
Trunc	Converte um valor do tipo real num valor do tipo inteiro, truncando seu valor
IntTostr	Converte um número numa String
IntToHex	Converte um número numa String com sua representação hexadecimal.
StrToInt	Converte uma String num número, suscitando uma exceção se a String não estiver correta
Val	Converte uma String num número
Str	Converte um número numa String, usando parâmetros de formatação
StrPas	Converte uma String terminada em nulo numa String estilo Pascal
StrPCopy	Converte (copia) uma String do estilo Pascal numa String terminada em zero
StrPLCopy	Converte (copia) uma parte da String ao estilo Pascal para uma String terminada em zero

2.4. Tipos de Dados definidos pelo Usuário

Juntamente com a noção de tipo, uma das grandes idéias introduzidas pela linguagem Pascal é a capacidade de definir novos tipos de dados num programa. Além de usarem os tipos de dados predefinidos, os programadores podem definir seus próprios tipos de dados por meio de *construtores de tipos*, tais como faixas secundárias, matrizes, registros, enumerações, ponteiros e conjuntos.

Estes tipos podem receber um nome para uso posterior ou ser aplicados a uma variável diretamente. Quando você dá um nome a um tipo, deve oferecer uma seção específica no código, como a seguinte:

```
type
  { subrange definition }
     uppercase = 'A'..'Z';
  { array definition }
     Temperatures = array [1..24] of Integer;
  { record definition }
     Data = record
       Dia: byte;
       Mes: byte;
       Ano: Integer;
    end;
 { enumerated type definition }
    Colors = (Red, Yellow, Green, Cyan, Blue, Violet);
 { set definition }
    Letters = set Of Char;
```

Construções semelhantes de definições de tipos podem ser usadas diretamente para definir uma variável sem uma definição de tipo explicita, como em:

```
var
```

```
   DecemberTemperature: array [1..31] of Byte;
   ColorCode: array [Red..Violet] of Word;
   Palette: set of Colors;
```

2.4.1. Faixas Secundárias

Um *tipo faixa secundária* define uma faixa de valores dentro da faixa de outro tipo (daí o nome faixa secundária*).* Você pode definir uma faixa secundária de números inteiros, de 1 a 10 ou de 100 a 1.000. Ou pode definir uma faixa secundária de caracteres, como em:

```
type
  Uppercase = 'A'..'Z';
```

Quando tiver definido uma faixa secundária, você poderá legalmente atribuir a ela um valor dentro dessa faixa.
Esta é válida: `UppLetter := 'F';`
Mas esta não é: `UppLetter := 'e';`

Escrever este código no Delphi 2 resulta em um erro de tempo de compilação, *Constant expression vialates subrange bounds.* Se ao invés disso, você escrever o seguinte:

```
var
   UppLetter: Uppercase;
   Letter: Char;
begin
   Letter := 'e';
   UppLetter := Letter;
end;
```

o Delphi compilará o código. Em tempo de execução, se você tiver habilitado a opção do compilador Range Checking, receberá uma mensagem de erro *Range check error.* Sugiro que você mantenha habilitada essa opção do compilador (veja a página Compiler da caixa de diálogo Project Options) enquanto estiver desenvolvendo um programa, e depois eventualmente desabilite-a para a compilação final. Isso torna o programa um pouco mais rápido, mas um pouco menos robusto. O mesmo é verdadeiro para outras opções de verificação de tempo de execução, tais como overflow e stack checking.

2.4.2. Enumerações

Tipos enumerados constituem outro tipo ordinal definido pelo usuário. Em vez de indicar uma faixa e um tipo existente, numa enumeração você lista todos valores possíveis para o tipo. Uma enumeração é uma lista de valores. Veja alguns exemplos:

```
type
  Cores = (Vermelho, Amarelo, Verde, Azul, Violeta);
```

Cada valor da lista tem uma *ordinalidade* associada, que se inicia em zero. Quando aplica a função Ord a um valor de um tipo enumerado, você obtém este valor baseado em zero. Por exemplo, Ord (Amarelo) retorna 1. No Delphi, há diversas propriedades que têm um valor enumerado. Por exemplo, o estilo borda de um formulário é definido da seguinte maneira:

```
TFormBorderStyle = (bsNone, bsSingle, bsSizeable, bsDialog, bsSizeToolWin,
bsToolWindow);
```

Quando o valor de uma propriedade é uma enumeração, usualmente você pode escolher da lista de valores exibida no Object Inspector:

2.4.3. Conjuntos

Um *tipo conjunto* indica o conjunto potência de um tipo ordinal, muito freqüentemente uma enumeração, uma vez que o tipo básico não pode ter mais de 256 possíveis valores em sua faixa. Cada conjunto pode conter nenhum, um, mais de um ou todos os valores dentro da faixa do tipo ordinal.

```
Type
  Uppercase = 'A'..'Z';
```

```
  Letras = set of Uppercase;
```

Agora posso definir uma variável deste tipo e atribuir a ela alguns valores do tipo original. Para indicar alguns valores num conjunto, escreva uma lista separada por vírgulas, entre colchetes. Quando tiver definido uma variável tal como:

```
var
   MinhasLetras: Letras;
```

Você poderá definir seu valor com as seguintes instruções (atribuindo respectivamente diversos valores, um único valor e um valor vazio):

```
MinhasLetras := ['A', 'B', 'C'];
MinhasLetras := ['K'];
MinhasLetras := [];
```

No Delphi, freqüentemente são usados conjuntos para indicar flags não-exclusivas. Por exemplo, as duas linhas seguintes declaram uma enumeração de possíveis ícones para a borda de uma janela e o conjunto correspondente:

```
type
  TBorderIcon = (bisystemMenu, biMinimize, biMaximize, biHelp);
  TBorderIcons = set of TBorderIcon;
```

De fato, uma janela pode ter nenhum desses ícones, um ícone, dois ícones, ou todos eles. Quando está trabalhando com o Object inspector, você pode oferecer os valores de um conjunto escrevendo a lista apropriada entre colchetes ou expandindo a seleção e ativando/desativando a presença de cada valor.

2.4.4. Matrizes

Uma matriz define um número fixo de elementos de um tipo específico. Por exemplo, você pode definir um grupo de 24 números inteiros com este código:

```
type
  TemperaturasDia = array [1..24] of Integer;
```

Na definição de matriz, você precisa usar duas constantes de um tipo ordinal para especificar os índices válidos da matriz. Desde que você especifique tanto o índice superior como o inferior da matriz.

No caso anterior, você pode definir o valor de uma variável TempDia do tipo TemperaturasDia como é mostrado a seguir:

```
var
  TempDia: TemperaturasDia;

Begin
    TempDia [1] := 34;
    TempDia [2] := 32;
     . . .
    TempDia [24] :=36;
end;
```

Uma matriz pode ter mais de uma dimensão, como nos seguintes exemplos:

```
type
  TemperaturasMes = array [1..24, 1..31] of Integer;
  TemperaturasAno = array [1..24, 1..31, 1..12] of Integer;
```

Portanto você poderia escrever:

```
var
   TempMes : TemperaturasMes;
```

```
Begin
   TempMes[1, 1] := 34;
   TempMes[1, 5] := 30;
   TempMes[3, 2] := 31;
End;
```

Você também pode definir matrizes baseadas em zero - matrizes com o limite inferior definido em zero. Geralmente, o uso de limites mais lógicos é uma vantagem, uma vez que você não precisa usar o índice 2 para acessar o terceiro item, e assim por diante.

2.4.5. Registros

Um *tipo registro* define uma coleção fixa de elementos de diferentes tipos. Cada elemento, ou *campo* tem seu próprio tipo. A definição de um tipo registro lista todos estes campos, dando a cada u deles um nome que você usará para acessá-lo. Eis um exemplo da definição de um tipo registro e uma variável desse tipo e o uso dessa variável:

```
Type
   Data = record
       Ano: Integer;
       Mes: Byte;
       Dia: Byte;
  end;

var
   Aniver: Data;

begin
    Aniver.Ano := 1995;
    Aniver.Mes = 2;
    Aniver.Dia := 14;
end;
```

2.5. Strings Pascal

Em Pascal, um tipo String é uma seqüência de caracteres com um contador de extensão que pode mudar dinamicamente. Cada String tem um tamanho fixo (o qual, por padrão, é 255), não obstante usualmente ela mantenha um número menor de caracteres. Um exemplo é:

```
var
    Nome: String;
    Titulo: String [50];
```

Nome é uma String de 255 caracteres. *Título* sustenta um máximo de 50 caracteres. Strings tradicionais do Pascal são limitados a 255 caracteres, e este é um problema real.

Como se pode ver a partir de uso de colchetes, as strings são semelhantes às matrizes. De fato, uma string é quase uma matriz de caracteres. Isto é demonstrado pelo fato de você poder escrever:

```
PrimeiroCaracter := Nome [1];
```

para acessar o primeiro caractere da String Nome. Há uma série de funções que podem ser usadas para operar sobre strings, como se pode ver na Tabela 2.3. Em particular, ao usar strings Pascal, você pode facilmente adicionar duas strings usando o sinal de mais:

```
StringFinal = PrimString + '  ' + SegString;
```

Esta expressão funde as duas strings, adicionando um caractere em branco (um espaço) entre elas.

2.6. Strings Pascal Longas

Para superar as limitações das strings tradicionais do Pascal, o Delphi apresenta suporte para strings longas. Na realidade existem dois tipos de String:

· Short String Corresponde a strings típicas do Pascal, como descrito na seção anterior. Essas strings têm uma limitação de 255 caracteres e correspondem às strings encontradas na versão 16 bits do Delphi.

· ANSIString Corresponde ao novo comprimento variável, strings long. Essas strings são alocadas dinamicamente e são praticamente ilimitadas.

Na versão 32 bits do Delphi, se você simplesmente utilizar o tipo de dado String, você terá tanto strings short como strings ANSI, dependendo do valor da nova diretiva de compilador $H. O default e $H+, que corresponde a strings long (ANSIString).

2.7. Conversões de Strings

Existe um par de funções de conversão que será bastante útil. StrPas converte uma String terminada em nulo em uma String short Pascal. StrPCopy faz a conversão inversa.

2.8. Estilo de Codificação

Antes começar a escrever instruções em linguagem Pascal, é importante destacar alguns dos elementos do estilo de codificação Pascal. Há alguns princípios que você deve conhecer em relação a comentários, uso de letras maiúsculas, espaços e a chamada "pretty Printing".

2.8.1. Comentários

Em Pascal, os comentários são colocados ou entre chaves ou entre parênteses seguidos de um asterisco:

```
{isto e um comentário}
(* este e outro comentário *)
// este e um comentário até o final da linha
```

Dispor de três formas de comentário pode ser útil para fazer comentários aninhados. Se comentar diversas linhas de código-fonte, e essas linhas possuírem comentários reais, você não poderá utilizar o mesmo identificador de comentário.

```
{
   ... código
   ... código
   { comentário, criando problemas, uma vez que o próximo caractere
      de fechar-chaves é relacionado à primeira chave acima }
   ... código
   ... código
 }
```

Com um segundo identificador de comentário, você pode escrever o seguinte código, que está correto:

```
{ ... código
    //este comentário está OK
   ... código }
```

Note que, se a chave de abertura ({) ou o parêntese-asterisco for seguido de um cifrão ($), eles se tornam uma diretiva de compilação, como em: {$X+}

2.8.2. Uso de Letras Maiúsculas

O compilador Pascal ignora letras maiúsculas. Por conseguinte, os identificadores Myname, MyName, myname, myName e MYNAME são todos exatamente equivalentes.

Uma abordagem comum é colocar em maiúsculo somente a primeira letra de cada identificador. Quando um identificador for composto de diversas palavras consecutivas (você não pode inserir um espaço num identificador), toda primeira letra de uma palavra deve ser colocada em maiúsculo, como em: `MeuIdentificadorLongo`

O uso coerente de maiúsculas/minúsculas não é imposto pelo compilador, mas é um bom hábito assumi-lo.

2.8.3. Espaços em Branco

Outros elementos completamente ignorados pelo compilador são a quantidade de espaços, novas linhas e espaços de tabulação que você deve adicionar ao código fonte. Todos estes elementos são coletivamente conhecidos como espaços em branco. Os espaços em branco são usados para melhorar a legibilidade do código; eles não afetam a compilação.

Novamente, não existem regras fixas para o uso de espaços e instruções de múltiplas linhas; apenas algumas regras empíricas:

· O editor Delphi tem uma linha vertical que você pode colocar após 60 ou 70 caracteres. Se você usar esta linha e evitar ultrapassar este limite, seu código-fonte terá uma aparência melhor quando você o imprimir.

· Quando uma função ou procedimento tem diversos parâmetros, uma prática comum consiste em colocar os parâmetros em linhas diferentes.

· Você pode deixar uma linha completamente em branco (vazia) antes de um comentário ou dividir uma peça longa de código em partes menores.

· Use espaços em branco para separar os parâmetros de uma chamada de função. Você pode também separar os operandos de uma expressão.

A última sugestão sobre o uso de espaços em branco relaciona-se ao estilo de formatação pico da linguagem Pascal, conhecido como *pretty-Printing.*

2.8.4. Pretty-Printing

A formatação do código-fonte em Pascal usualmente segue uma abordagem padrão, conhecida como *pretty-Printing.* Sua regra e simples: cada vez que você precisar escrever uma instrução composta, recue-a dois espaços à direita do restante da instrução atual. Uma instrução composta dentro de outra instrução composta e recuada quatro espaços, e assim por diante:

```
if ... then
    instrução;
if ... then
    begin
      instrução1;
      instrução2;
    end;
if ... then
    begin
      if ... then
          instruçãol;
      instrução2;
    end;
```

Obviamente, qualquer uma dessas convenções é apenas uma sugestão para tornar o código mais legível a outros programadores, e ela é completamente ignorada pelo compilador.

2.8.5. Destaque da Sintaxe

Para tornar mais fácil ler e escrever código Delphi, o editor Delphi tem uma característica chamada *destaque em cores da sintaxe*. Dependendo do significado na linguagem Pascal das palavras que você digita no

editor, elas são exibidas usando diferentes cores. Por padrão, as palavras-chave estão em negrito, as strings e os comentários estão em cores e assim por diante.

As palavras reservadas, comentários e strings provavelmente são os três elementos que mais se beneficiam desta característica. Você pode ver num relance uma palavra-chave escrita erroneamente uma String não adequadamente terminada, e a extensão de um comentário de múltiplas linhas.

Você pode personalizar facilmente as configurações de destaque da sintaxe usando a página Editor Colors da caixa de diálogo Environment Options.

2.9. Instruções Pascal

Assim que tiver definido alguns identificadores, você poderá usá-los em instruções e nas expressões que fazem parte de algumas instruções. Em Pascal, existem diversas instruções e expressões diferentes, Vamos analisar as expressões e operações primeiro.

2.9.1. Expressões e Operadores

Uma expressão é qualquer combinação válida de constantes, variáveis, valores literais, operadores e resultados de funções. Não há uma regra geral para criar expressões, uma vez que elas dependem principalmente dos operadores quando são usados. Há operadores lógicos, aritméticos, relacionais e de conjunto, além de alguns outros. As expressões também são usadas para determinar o valor a ser atribuído a uma variável, para computar o parâmetro de uma função ou procedimento, ou para testar uma condição. As expressões podem incluir chamadas de funções.

As expressões são para somente-leitura. Você pode usar o resultado de uma expressão, mas não pode atribuir um valor a ela. Em outras palavras, uma expressão pode aparecer no lado direito de uma atribuição, mas não no lado esquerdo. As expressões também podem ser passadas a parâmetros de valor de procedimentos e funções, não a parâmetros de referencia (`var`).

Tabela 2.3: Operadores da linguagem Pascal

Operador	Propósito
Operadores Unários	
@	Endereço de (retorna um ponteiro)
not	"Não" booleano ou bit voltado para "não"
Operadores Multiplicativos e de Direção de Bit	
*	Multiplicação aritmética ou intersecção de conjuntos
/	Divisão de tipo real
div	Divisão de tipo inteiro
mod	Módulo (o resto de uma divisão de números inteiros)
as	Typecast seguro quanto ao tipo (RTTI)
and	"E" booleano e bit voltado para "e"
shl	Deslocamento de bits à esquerda
Shr	Deslocamento de bits à direita
Operadores Aditivos	
+	Adição, união de conjuntos, concatenação de strings, valor positivo, ou adição de compensação (offset)
-	Subtração, diferença de conjuntos, valor negativo, ou subtração de compensação (offset)
Or	"Ou" booleano ou direcionamento de bit para "ou"
Xor	"O" booleano ou direcionamento de bit para "ou" exclusivo
Operadores Relacionais e de Comparação	
=	Testar se e igual
<>	Testar se não e igual

<	Testar se e menor que
>	Testar se e maior que
<=	Testar se e menor ou igual ou se é subconjunto de um conjunto
>=	Testar se e maior ou igual ou se é superconjunto de um conjunto
in	Testar se e membro de
Is	Testar se o tipo é compatível (RTTI)

Note que alguns dos operadores mais comuns têm diferentes significados com diferentes tipos de dados. Por exemplo, o operador + pode ser usado para adicionar dois números, concatenar duas strings, fazer a união de dois conjuntos.

Outro operador estranho é **div**. Em Pascal, você pode dividir dois números quaisquer (reais ou inteiros) com o operador /, e invariavelmente você obterá um resultado do tipo real Se você precisar dividir dois números inteiros e quiser um resultado inteiro, use o operador div. (A propósito, div e mais rápido que /.)

2.9.2. Operadores de Conjuntos

Os operadores de conjuntos incluem:

União de conjuntos (+)
Diferença (-)
Intersecção (*)
Teste de pertinência de conjuntos (in)
Operadores relacionais

Para adicionar um elemento a um conjunto, você pode fazer a união do conjunto com outro que tenha somente o elemento de que precisa. Eis um exemplo Delphi relacionado a estilos de fonte:

```
Style := Style + [fsBold];
Style := Style + [fsBold, fsItalic] - [fsUnderline];
```

Como alternativa, você pode usar os procedimentos padrões Include e Exclude, os quais são muito mais eficientes: `Include (Style, fsBold);`

2.10. Instruções Condicionais Pascal

Uma instrução condicional é usada para executar uma ou nenhuma das instruções que a compõe. Há duas opções básicas de instruções condicionais: instruções if e instruções case.

2.10.1. Instruções IF

As instruções if podem ser usadas para executar uma instrução somente se certa condição for satisfeita (if-then), ou para escolher entre duas instruções diferentes (if-then-else). A condição é descrita com uma expressão booleana.

Examinaremos um exemplo Delphi simples para mostrar como escrever instruções de condição simples. Primeiro, crie um novo aplicativo em branco e inclua duas caixas de verificação e quatro botões no formulário.

Não mude os nomes dos botões ou das caixas de verificação, mas dê um clique duplo sobre cada um dos botões para adicionar um manipulador de eventos OnClick. Eis uma instrução if simples para o primeiro botão:

```
Procedure TForml.Button1c1ick(sender: TObject);
begin
   {instrução if simples}
   if CheckBoxl.checked then ShowMessage ('CheckBoxl está marcada');
end;
```

Quando você der um clique sobre o botão, se a primeira caixa de verificação tiver em si uma marca de verificação, o programa mostrará uma mensagem simples numa pequena janela.

Se você der um clique sobre o botão e nada acontecer, isso significa que a caixa de verificação não foi marcada. Em geral, é melhor tornar isto mais explícito, como neste código para o segando botão, que usa uma instrução if-then-else.

```
Procedure Tforml.Button2click(sender: TObject);
begin
   { instrução if-then-else }
   if CheckBox2.Checked then ShowMessage ('CheckBox2 está marcada')
   else ShowMessage ('CheckBox2 NÃO está marcada');
end;
```

Note que não se pode ter um ponto-e-vírqula após a primeira instrução e antes da palavra-chave else, ou o compilador emitirá um erro de sintaxe.

Instruções if podem ser muito complexas. A condição pode ser transformada numa série de condições, ou as instruções if podem aninhar uma segunda instrução if. Os dois últimos botões do exemplo demonstram estes casos.

```
Procedure TForml.Button3Click(Sender: TObject);
begin
   {  instrução com dupla condição  }
    if CheckBoxl.Checked and CheckBox2.Checked then
      ShowMessage ('Ambas as caixas estão marcadas');
end;

Procedure TForml.Button4Click(Sender: TObject);
begin
   {instrução if composta}
    if CheckBoxl.Checked then
       if CheckBox2.Checked then
           ShowMessage ( 'CheckBoxl e 2 estão marcados' )
       else ShowMessage ( 'Somente CheckBoxl está marcado' )
    else ShowMessage ( 'Checkboxl não está marcado' );
end;
```

2.10.2. Instruções Case

Se suas instruções if se tornaram muito complexas, você pode substituí-las por instruções case. Uma instrução case consiste em uma expressão usada para selecionar um valor em uma lista de possíveis valores ou faixas de valores. Estes valores são constantes e devem ser únicos e de tipo ordinal, Finalmente, pode haver uma instrução else que será executada se nenhum dos rótulos responder ao valor do seletor. Eis dois exemplos simples:

```
case Numero of
  1: Texto := 'Um';
  2: Texto := 'Dois';
  3: Texto := 'Três';
end;

begin
  case MeuChar of
      '+'  :  Texto := ' sinal mais ';
      '-'  :  Texto := ' Sinai menos ';
      '0'..'9' : Texto := ' Número ';
      'a'..'z' : Texto := 'Caractere minúsculo';
      'A'..'Z' : Texto := 'caractere maiúsculo';
      else Texto :=  'caractere desconhecido';
  end;
end;
```

2.11. Loops Pascal

A linguagem Pascal tem as instruções repetitivas típicas da maioria das linguagens de programação, inclusive as instruções for, while e repeat.

2.11.1. Instruções FOR

O loop for na linguagem Pascal baseia-se estritamente num contador, o qual pode ser aumentado ou diminuído cada vez que o loop for executado. Eis um exemplo simples de um loop for usado para adicionar os dez primeiros números inteiros.

```
begin
   K := 0;
   for i := 1 to 10 do
       k := k + i;
end;
```

A mesma instrução for poderia ter sido escrita usando-se o contador inverso:

```
begin
   K := 0;
   for i := 10 Downto 1 do
       K := K + i;
end;
```

2.11.2. Instruções While e Repeat

A diferença entre o loop while-do e o loop repeat-until é que o código da instrução repeat sempre é executado pelo menos uma vez. Você pode entender facilmente o porquê disso ao observar um exemplo simples:

```
While I < 100 and J < 100 do
begin
   {use I e J para computar algo...}
   I := I + 1;
   J := J + 1;
end;

Repeat
   {use I computar algo...}
   I := I + 1;
until I >= 100;
```

Se o valor inicial de I ou J for maior que 100, as instruções dentro do loop repeat-until serão executadas de qualquer forma.

Para explorar os detalhes dos loops, vamos examinar um pequeno exemplo do Delphi Este exemplo, denominado LOOPS, realça a diferença entre um loop com um contador fixo e loop com um contador quase aleatório.

Inicie com um projeto em branco, coloque uma caixa de lista e dois botões no formulário principal, e dê aos botões um nome apropriado ("ButtonFor" e "ButtonWhile"), removendo palavra *button* das legendas.

Agora podemos adicionar algum código aos eventos OnClick dos dois botões. O primeiro botão tem um loop for simples para exibir uma lista de números.

Antes de executar este loop, o qual adiciona uma série de strings à propriedade item da caixa de listagem, você precisa limpar o conteúdo da própria caixa de listagem:

```
Procedure TForml.ForButtonClick(Sender TObject);
var
```

```
I: Integer
begin
  Listboxl.Items.clear;
  for I := 1 to 20 do
      Listboxl.Items.Add ('String ' + IntTostr(I));
end;
```

O código associado ao segundo botão é ligeiramente mais complexo. Neste caso, o loop while baseado num contador, o qual é aumentado aleatoriamente. Para realizar isto, chamei procedimento Randomize, o qual restabelece o gerador de números aleatórios, e a função Random com um valor limite igual a 100. O resultado desta função é um número entre 0 e 99, escolhido aleatoriamente.

```
Procedure TForml.WhileButtonClick (Sender : Tobject);
var
  I: Integer;
begin
  ListBoxl.Items.Clear;
  Randomize;
  I := 0;
  while I < 1000 do
      begin
        I := I + Random (100);
        Listboxl.Items.Add ('Numero Randômico: ' + IntTostr (I));
      end;
end;
```

Cada vez que você der um clique sobre o segundo botão, os números serão diferentes, porque eles dependem do gerador de números randômicos.

Você pode alterar o fluxo padrão de uma execução de loop utilizando os procedimentos de sistema `Break` e `Continue` O primeiro interrompe o loop; o segundo é utilizado para pular diretamente para o teste do loop ou incremento do contador, continuando com a iteração seguinte ou com o loop (a menos que a condição seja zero ou que o contador tenha atingido seu valor mais alto). Dois outros procedimentos de sistema, `Exit` e `Halt`, permitem-lhe sair da função atual de procedimento ou finalizar o programa.

2.12. A Instrução With

A instrução with nada mais é do que taquigrafia. Quando você precisa referir-se a um registro (ou a um objeto), em vez de repetir seu nome todas as vezes, você pode usar uma instrução with. Por exemplo, para alterar algumas propriedades de Button1 você pode escrever:

```
With Button1 do
  begin
      Caption := 'Teste';
      Left := 10;
      Top := 20;
  end;
```

Quando você trabalha com componentes ou classes em geral, a instrução with permite-Ihe economizar código, especialmente para campos aninhados.

Quando você estiver escrevendo um código complexo, a instrução with pode ser eficaz, mas há um inconveniente. Ela pode tornar o código menos legível, particularmente quando você estiver trabalhando com objetos diferentes que tenham propriedades semelhantes ou correspondentes.

2.13. Procedimentos e Funções Pascal

Outra idéia importante sublinhada pela linguagem Pascal é o conceito da sub-rotina. Na linguagem Pascal, as sub-rotinas podem assumir duas diferentes formas: procedimentos e funções. A única diferença real entre as duas construções é que as funções têm um valor de retorno, enquanto os procedimentos não retornam um

valor. Eis as definições de um procedimento e duas versões da mesma função, usando uma sintaxe ligeiramente diferente:

```
Procedure Ola;
begin
  ShowMessage ('Olá Mundo');
end;

Function Duplo : Integer;
begin
  Duplo := 100;
end;

Function Duplo2 (valor: Integer): Integer;
begin
  Result := Valor * 2;
end;
```

Assim que estas sub-rotinas tiverem sido definidas, você poderá chamá-las da seguinte maneira:

```
begin
  Ola;
  X := Duplo;
  Z := Duplo2 (X);
end;
```

2.13.1. Parâmetros de Referência

Tanto os procedimentos como as funções permitem a passagem de parâmetros por valor ou por referência. Passar um parâmetro por referência significa que seu valor não será copiado na pilha do parâmetro formal da sub-rotina (evitar uma cópia freqüentemente significa que o programa economizará algum tempo). Em vez disso, o programa refere-se ao valor original também no código da sub-rotina. Isto permite que o procedimento ou função mude o valor do parâmetro. A passagem do parâmetro por referência é expressa pela palavra-chave **`var`**. Exemplo:

```
Procedure Dobra0Valor (var Valor: Integer) ;
begin
  Valor = Valor * 2;
end;
```

Neste caso, o parâmetro é usado tanto para passar um valor ao procedimento como para retornar um novo valor ao código de chamada.

2.13.2. Parâmetros Constantes

Como uma alternativa a parâmetros de referência, você pode utilizar um parâmetro **const**. Uma vez que você não pode atribuir um novo valor a um parâmetro const dentro da rotina, o compilador pode otimizar a passagem de parâmetro, particularmente para strings ou registros grandes. O compilador pode escolher entre uma abordagem similar aos parâmetros de referência, mas o comportamento permanecerá similar aos parâmetros de valor, porque o valor interno não será afetado pela rotina. De fato, se você tentar compilar o (ingênuo) código a seguir, o Delphi exibirá um erro:

```
Function DobraOValor (const Valor: Integer) : Integer;
begin
  Valor := Valor * 2;   // Erro
  DobraOValor := Valor;
end;
```

2.13.3. Parâmetros de Abertura de Matrizes

Uma função ou Procedure do Pascal possui um conjunto de parâmetros definido. Entretanto, existe uma forma de passar uma quantidade genérica de parâmetros a uma função, Utilizando - a *matriz aberta.* Esse é um tipo especial de matriz que possui uma quantidade indefinida de valores, que pode ser útil para passagem de parâmetros. Por exemplo, essa é a definição da função Format do sistema.

```
Function Format (const Format: String; const Args: array of const): String;
```

O segundo parâmetro é um array aberto, que recebe um número indefinido de valores**.** De fato, você pode chamar esta função das seguintes formas:

```
N := 20;
S : =  'Total: ';
Labell.Caption := Format ('Total: %d', [N]) ;
Label2.Caption := Format ('Int: %d, Float: %f, [N, 12.4]);
Label3.Caption := Format ('%s %d', [S, N * 2]);
```

Observe que pode passar um parâmetro tanto como um valor constante, o valor de uma variável, ou até mesmo uma expressão. A função Format pode ser muito útil para construir strings de saídas complexas.

2.14. 0 Que é um Método?

Um método é um tipo especial de função ou procedimento que está relacionado ao tipo de dados, uma *classe.*

No Delphi, todas as vezes que manipulamos um evento ou mensagem, precisamos definir um método, o qual pode ser uma função ou um procedimento. Por esta razão, o termo *método* é usado para indicar funções e procedimentos em geral, embora isso nem sempre aconteça. Falando estritamente, somente as sub-rotinas relacionadas a uma classe são métodos.

Veja um exemplo de método vazio automaticamente adicionado pelo Delphi ao código-fonte de um formulário:

```
Procedure TForml.ButtonlClick(Sender: TObject);
begin

end;
```

2.15. Tratamento de exceções

Para criar aplicações robustas é necessário manter um controle sobre as exceções que podem ocorrer durante a execução do programa. Este mecanismo permite que a aplicação se recupere de erros ou finalize a execução de algum processo, sem a perda de dados ou recursos.

O tratamento de exceções permite ao programador manter um controle sobre os erros que podem acontecer e trata-los de forma consistente. Se um erro ocorrer em um procedimento que não possui tratamento de exceções então uma mensagem padrão de erro será exibida de acordo com o tipo da exceção, a execução será interrompida no ponto em que o erro ocorreu e o compilador voltará a executar a partir de um ponto seguro.

As exceções são classes definidas no Delphi para permitir o tratamento de erros. Estas classes herdam direto de TObject e ainda podem ser especializadas pelo programador em sub-classes. Algumas classes para o tratamento de exceções estão descritas abaixo:

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 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	Resultado de um cálculo com número real excedeu o limite
ErangeError	Valor excede o limite do tipo inteiro ao qual foi atribuída
Eunderflow	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

2.15.1. Levantando uma exceção

Você pode levantar uma exceção no seu código para indicar a ocorrência de um erro que deve ser tratado por quem fez a chamada. Por exemplo, suponha que você esteja implementando uma função de uma calculadora que efetuará as operações. Sabemos que a operação de um número por zero é inválida, portanto você pode levantar esta exceção dentro da sua função que calcula os valores. Veja exemplo a seguir:

```
function Divide (numerador, denominador: integer): real;
begin
  if denominador = 0 then
    raise EdivByZero.Create;
  result := numerador / denominador;
end;
```

2.15.2. Blocos protegidos

Um bloco protegido é uma seqüência de comandos delimitada por palavras-chave que indicam o tratamento da exceção. Qualquer erro que ocorrer neste bloco será tratado consistentemente.

Exemplo:

```
try
  x := x / y;     { x := Divide(x, y); usando a função acima }
except
  on E: EdivByZero do
  begin
    ShowMessage('Divisão de um número inteiro por zero.');
    Abort;
  end;
end;
```

2.15.3. Blocos de finalização

Um bloco de finalização é geralmente utilizado para liberar recursos que tenham sido alocados, pois estes blocos são sempre executados mesmo que ocorra uma exceção.

Exemplo:

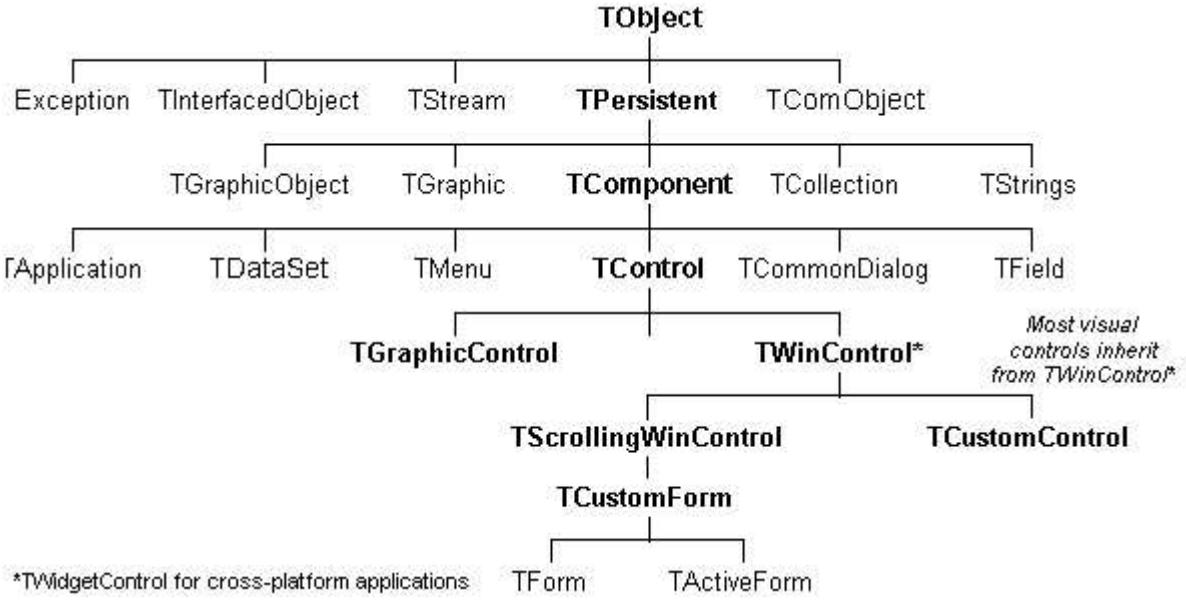
```
Bmp := TBitmap.Create;
try
  try
    Bmp.LoadFromFile('c:\teste.bmp');
  except
    on E: EFOpenError do
      raise Exception.Create('O arquivo não foi encontrado.');
    on E: Exception do
```

```
      raise Exception.Create('Não foi possível abrir o arquivo. ' +
                             E.Message);
  end;
finally
  Bmp.Free;
end;
```

3. BIBLIOTECA DE COMPONENTES (VCL E CLX)

3.1. A Base Conceitual

A biblioteca de componentes do Delphi/Kylix é organizada de forma hierárquica. Nesta hierarquia estão definidas as Bibliotecas de Componentes Visuais (Visual Component Library - VCL) e a Biblioteca de Componentes Independente de Plataforma (Component Library for Cross-Platform – CLX). Veja abaixo a hierarquia:



Para criar aplicativos independentes de plataforma, ou seja, que possam ser compilados e executados tanto no Linux quanto no Windows, é necessário que estes sejam criados utilizando-se componentes da CLX.

Todos os componentes e objetos utilizados no desenvolvimento dos aplicativos fazem parte desta biblioteca. E juntos eles formam uma hierarquia de classes. Como cada classe do sistema é uma subclasse do tipo de dados TObject, toda a hierarquia tem uma única raiz. Isto permite que você use TObject como substituto para qualquer tipo de classe do sistema.

3.1.1. A Hierarquia da VCL

A VCL define uma série de subclasses de TObject. Muitas destas classes são de fato subclasses de outras subclasses, formando uma hierarquia complexa. Você pode consultar os arquivos Help para verificar árvore hierárquica da VCL.

Você não precisa conhecer toda a hierarquia da VCL para poder trabalhar com o Delphi, a não ser que pretenda desenvolver componentes. Neste caso será necessário um conhecimento maior sobre a hierarquia.

Se você pretende desenvolver aplicativos para a plataforma Windows apenas, então é recomendável a utilização da VCL ao invés da CLX, por causa da eficiência na utilização de recursos do sistema.

3.1.2. A Hierarquia da CLX

Assim como a VCL, a CLX também define uma série de subclasses de TObject. A hierarquia da CLX se inicia logo abaixo de TControl, substituindo a classe TWinControl por TWidgetControl.

Se você pretende desenvolver aplicativos multi-plataforma então será necessário utilizar os componentes da CLX ao invés da VCL. Aliás, se pretende desenvolver para Linux então você terá que usar a CLX.

Você pode também desenvolver novos componentes para a CLX, mas para isso precisará entender profundamente os fundamentos desta biblioteca.

3.2. Componentes

Os componentes são os elementos centrais dos aplicativos Delphi/Kylix. Quando você escreve um programa, basicamente escolhe uma série de componentes e define suas interações.

Há diferentes tipos de componentes no Delphi. A maioria dos componentes está incluída na paleta de Componentes, mas alguns deles (inclusive TForm e TApplication) não estão. Tecnicamente, os componentes são subclasses da classe TComponent. Como tal, eles podem ser manipulados visualmente, uma vez que você define suas propriedades em tempo de execução.

Os componentes podem ser visuais ou não-visuais. Em tempo de projeto, um componente não-visual aparece no formulário como um pequeno ícone. Em tempo de execução, alguns desses componentes são visíveis (por exemplo, as caixas de diálogo padrões), e outros são invisíveis (como algumas conexões de bancos de dados). Os componentes não visuais freqüentemente gerenciam algo que é visual como, por exemplo, uma caixa de diálogo.

3.3. Controles

Os controles podem ser definidos como componentes visuais. Você pode colocar um controle num formulário em tempo de projeto e ver como ele se apresenta, e pode ver os controles no formulário em tempo de execução. Os controles são responsáveis pela maioria dos componentes, e os termos são freqüentemente usados como sinônimos - embora existam componentes que não são controles.

3.4. Propriedades

Propriedades são atributos de classes que determinam o status de um objeto, como, por exemplo, sua posição e sua aparência, e também seu comportamento. Para saber o valor de uma propriedade em tempo de projeto, ou para mudá-la, você pode usar o Object Inspector. Em tempo de execução, você pode acessar uma propriedade lendo-a ou escrevendo-a com algum código simples.

O Object Inspector lista somente as propriedades de tempo de projeto de um componente, e as propriedades de tempo de execução não estão incluídas. Para obter uma lista completa das propriedades, consulte os arquivos de Help do Delphi.

Você pode estabelecer uma propriedade indicando um novo valor ou pode ler seu valor atual. Porém, há propriedades de somente-leitura e (poucas) propriedades de somente escrita. Ambos os tipos de propriedades usualmente estão disponíveis em tempo de execução.

Tabela 3.1: Algumas propriedades disponíveis para a maioria dos componentes.

Propriedade	Disponível para	Descrição
Align	Todos os controles	Determina como o controle é alinhado em sua área de controle de origem
BoundsRect	Todos os controles	Indica o retângulo demarcador do controle.
Caption	Todos os controles	A legenda do controle.
ComponentCount	Todos os componentes	O número de componentes possuídos pelo atual.
ComponentIndex	Todos os componentes	Indica a posição do componente na lista de componentes do proprietário.
Components	Todos os componentes	Uma matriz dos componentes possuídos pelo atual.
ControlCount	Todos os controles	O número de controles que são filhos do atual.
Controls	Todos os controles	Uma matriz dos controles que são filhos do atual
Color	Muitos objetos e componentes	Indica a cor da superfície, do fundo, ou a cor atual.
Ctrl3D	A maioria dos componentes	Determina se o controle tem uma aparência tridimensional
Cursor	Todos os controles	O cursor usado quando o ponteiro do mouse está

		sobre o controle.
DragCursor	A maioria dos controles	O cursor usado para indicar que o controle aceita ser arrastado.
DragMode	A maioria dos controles	Determina o comportamento "arrastar-e-soltar" do controle como componente inicial para uma operação de arrastar.
Enabled	Todos os controles e alguns outros componentes	Determina se o controle está ativo ou inativo (ou acinzentado).
Font	Todos os controles	Determina a fonte do texto exibido dentro do componente.
Handle	Todos os controles	O manipulador da janela usado pelo sistema.
Height	Todos os controles e alguns outros componentes	O tamanho vertical do controle.
HelpContext	Todos os controles e os componentes da caixa de diálogo	Um número contextual usado para chamar a ajuda contextual automaticamente.
Hint	Todos os controles	A string usada para exibir dicas instatâneas para o controle.
Left	Todos os controles	A coordenada horizontal do canto superior esquerdo do componente.
Name	Todos os componentes	O nome único do componente, o qual pode ser usado no código-fonte.
Owner	Todos os componentes	Indica o componente proprietário.
Parent	Todos os controles	Indica o controle de origem (pai).
ParentColor	Muitos objetos e componentes	Determina se o componente deve utilizar sua própria propriedade Color ou a do componente-pai.
ParentCtl3D	Maioria dos componentes	Determina se o componente deve utilizar sua própria propriedade Ctrl3D ou a do componente-pai.
ParentFont	Todos os controles	Determina se o componente deve usar sua própria propriedade Font ou a do componente de origem (pai).
ParentShowHint	Todos os controles	Determina se o componente deve usar sua própria propriedade ShowHint ou a do controle de origem (pai).
PopupMenu	Todos os controles	Indica o menu suspenso a ser usado quando o usuário der um clique sobre o controle com o botão esquerdo do mouse.
ShowHint	Todos os controles	Determina se as dicas estão ativadas.
Showing	Todos os controles	Determina se o controle está sendo mostrado atualmente na tela, ou seja, se está visível, quando seu controle de origem (pai) é mostrado.
TabOrder	Todos os controles (exceto TForm)	Determina a ordem de tabulação deste controle.
TabStop	Todos os controles (exceto TForm)	Determina se o usuário pode tabular para este controle.
Tag	Todos os componentes	Uma long integer disponível para armazenar dados personalizados.
Top	Todos os controles	A coordenada vertical do canto superior esquerdo do componente.
Visible	Todos os controles e alguns outros componentes	Determina se o controle é visível.
Width	Todos os controles e alguns outros componentes	O tamanho horizontal do controle.

3.4.1. A Propriedade Name

Todo componente do Delphi pode ter um nome. Se você der um nome a um componente, o nome deve ser único dentro do componente proprietário, geralmente um formulário. Isto significa que um aplicativo pode ter dois formulários diferentes em que cada um tenha um componente com o mesmo nome.

O Delphi usa o nome do componente para criar o nome padrão do método relacionado a seus eventos. Se você tiver um componente Button1, seu evento OnClick será conectado a um método Button1Click, a menos que você especifique um nome diferente. Se posteriormente você mudar o nome do componente, o Delphi modificará os nomes dos métodos relacionados de acordo.

3.4.2. Propriedades Relacionadas ao Tamanho e á Posição

Outras propriedades importantes, comuns à maioria dos componentes, são aquelas relacionadas ao tamanho e à posição. A posição de um componente é definida por suas propriedades Left e Top, e seu tamanho (disponível somente para os controles) é determinado pelas propriedades Height e Width.

A posição de um componente sempre se relaciona à área do cliente de seu componente de origem (pai, o qual é o componente indicado por sua propriedade Parent). Para um formulário, a área do cliente é a superfície incluída em suas fronteiras (mas sem as próprias bordas).

Observe que se você colocar um painel num formulário, e um botão no painel, as coordenadas do botão se relacionarão ao painel, e não ao formulário. Neste caso, o componente de origem (pai) do botão é o painel.

3.4.3. As propriedades Enabled, Visible e Showing

Há duas propriedades básicas que você pode usar para permitir que o usuário interaja com um componente existente. A mais simples é a propriedade *Enabled*. Quando um componente é desativado (quando Enabled é colocado em False) ele se torna acinzentado, e não responde aos comandos do usuário. Isto só é percebido em tempo de execução.

Você pode, também, ocultar completamente um componente definindo a propriedade Visible em False. Ou ocultar um grupo de componentes de uma só vez. Para isto você deve definir a propriedade Visible do contêiner destes componentes (por exemplo, um painel) para False. Isto também só é percebido em tempo de execução.

Para saber se um controle está visível ao usuário você pode utilizar a propriedade Showing. Se o componente estiver visível, seu controle de origem (pai) também deve estar visível.

3.4.4. A Propriedade Tag

Esta propriedade é meramente uma localização extra de memória, presente em cada classe de componentes, onde você pode armazenar valores personalizados. O tipo de informação armazenado e a maneira pela qual ele é usado dependem exclusivamente de você.

3.5. Métodos de Componentes

Os métodos de componentes são exatamente iguais a quaisquer outros métodos. Há procedimentos e funções relacionados a um objeto que você pode chamar para executar a ação correspondente.

Tabela 3.2: Alguns métodos disponíveis para a maioria dos componentes ou controles

Método	Disponível para	Descrição
BeginDrag	Todos os controles	Inicia o arrasto manual.
BringToFront	Todos os controles	Coloca o componente na frente de todos os outros.
CanFocus	Todos os controle	Determina se o controle pode receber o foco.
ClientToScreen	Todos os controles	Traduz coordenadas de tela.

ContainsControl	Todos os controles ajanelados	Determina se certo controle é contido pelo atual.
Create	Todos os objetos e componentes	Cria uma nova instância.
Destroy	Todos os objetos e componentes	Destrói a instância (se você tiver usado Create para criar o objeto, é melhor usar o método Free em vez de Destroy.
Dragging	Todos os controles	Indica se os controles estão sendo arrastados.
EndDrag	Todos os controles	Termina manualmente o arrasto.
FindComponent	Todos os componentes	Retorna o componente na propriedade da matriz Components que tenha determinado nome.
Focused	Todos os controles ajanelados	Determina se o controle tem o foco.
Free	Todos os objetos e componentes (não sugeridos para os formulários)	Apaga a instancia e a memória associada (os formulários devem usar, em vez disso, o método Release).
GetTextBuf	Todos os controles	Recupera o texto ou a legenda do controle.
GetTextLen	Todos os controles	Retorna a extensão do texto ou legenda do controle.
HandleAllocated	Todos os controles	Retorna True se o manipulador do controle existir.
HandleNeeded	Todos os controles	Cria um manipulador se ele ainda não existir.
Hide	Todos os controles	Torna o controle não-visível.
InsertComponent	Todos os componentes	Adiciona um novo elemento à lista de componentes possuídos.
InsertControl	Todos os controles	Adiciona um novo elemento à lista de controles que são filhos do atual.
Invalidate	Todos os controles	Força um redesenho do controle.
RemoveComponent	Todos os componentes	Remove um componente da lista Components.
ScaleBy	Todos os controles	Gradua o controle em determinada porcentagem.
ScreenToClient	Todos os controles	Traduz coordenadas de tela.
ScrollBy	Todos os controles	Rola o conteúdo do controle.
SendToBack	Todos os controles	Coloca o componente atrás de todos os outros.
SetBounds	Todos os controles	Muda a posição e o tamanho do controle (mais rápido do que acessar as propriedades relacionadas uma a uma).
SetFocus	Todos os controles	Coloca o foco de entrada no controle.
SetTextbuf	Todos os controles	Define o texto ou legenda do controle.
Show	Todos os controles	Torna o controle visível.
Update	Todos os controles	Redesenha imediatamente o controle, mas apenas se uma operação de redesenho tiver sido solicitada (ou seja, após uma chamada ao método Invalidate).

3.6. Eventos de Componentes

Quando um usuário pratica uma ação sobre um componente, como, por exemplo, dando um clique sobre ele, o componente gera um evento. Outras vezes, eventos são gerados pelo sistema como resposta à chamada de um método ou uma mudança de propriedade nesse componente (ou até mesmo um componente diferente). Por exemplo, se você definir o foco sobre um componente, o componente que tem atualmente o foco perde-o, disparando o evento correspondente.

Tabela 3.3: Alguns eventos disponíveis à maioria dos componentes.

Evento	Disponível para	Descrição
OnChange	Muitos objetos e componentes	Ocorre quando o objeto (ou seu conteúdo) muda.
OnClick	Muitos controles	Ocorre quando se dá um clique com o botão esquerdo do mouse sobre o componente.
OnDblClick	Muitos controles	Ocorre quando o usuário dá um clique duplo com

		o mouse sobre o componente.
OnDragDrop	Muitos controles	Ocorre quando uma operação de arrastar termina sobre o componente.
OnDragOver	Muitos controles	Ocorre quando o usuário está arrastando sobre o componente.
OnEndDrag	Muitos controles	Ocorre quando a ação de arrastar é encerrada e enviada ao componente que iniciou a operação de arrastar.
OnEnter	Todos os controles ajanelados	ocorre quando o componente é ativado, ou seja, quando o componente recebe o foco.
OnExit	Todos os controles ajanelados	Ocorre quando o componente perde o foco.
OnKeyDown	Muitos controles	Ocorre quando o usuário pressiona uma tecla e é enviado ao componente que detém o foco.
OnKeyPress	Muitos controles	Ocorre quando o usuário pressiona uma tecla e é enviado ao componente que detém o foco.
OnKeyUp	Muitos controles	Ocorre quando o usuário solta uma tecla e é enviado ao componente que detém o foco.
OnMouseDown	Muitos controles	Ocorre quando o usuário pressiona um dos botões do mouse e geralmente é enviado ao componente sob o cursor do mouse.
OnMouseMove	Muitos controles	Ocorre quando o usuário move o mouse sobre um componente.
OnMouseUp	Muitos controles	Ocorre quando o usuário solta um dos botões do mouse.
OnStartDrag	Muitos controles	Ocorre quando o usuário inicia o arrasto; é enviado para o componente que inicia a operação de arrastar.

4. CRIANDO APLICATIVOS DE BANCOS DE DADOS

4.1. Bancos de Dados e Tabelas

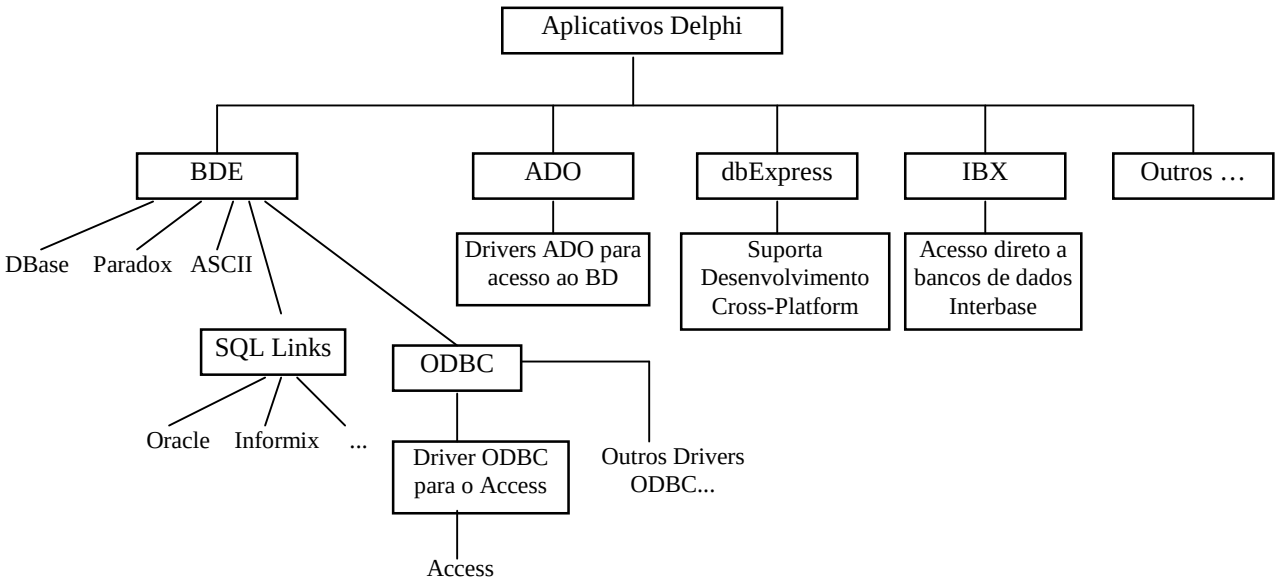
Um Banco de dados pode ser visto como um conjunto de arquivos, freqüentemente armazenados em um único diretório.

No Delphi, você sempre se refere a um banco de dados pelo seu nome ou por meio de um "alias", que é uma espécie de apelido do banco de dados, mas essa referência pode ser a um arquivo de banco de dados ou a um diretório que contém os arquivos com as tabelas.

O Delphi pode usar tabelas dBASE ou Paradox e acessar bancos de dados servidores SQL ou bancos de dados em outros formatos através do padrão ODBC da Microsoft (Open Database Connectivity).

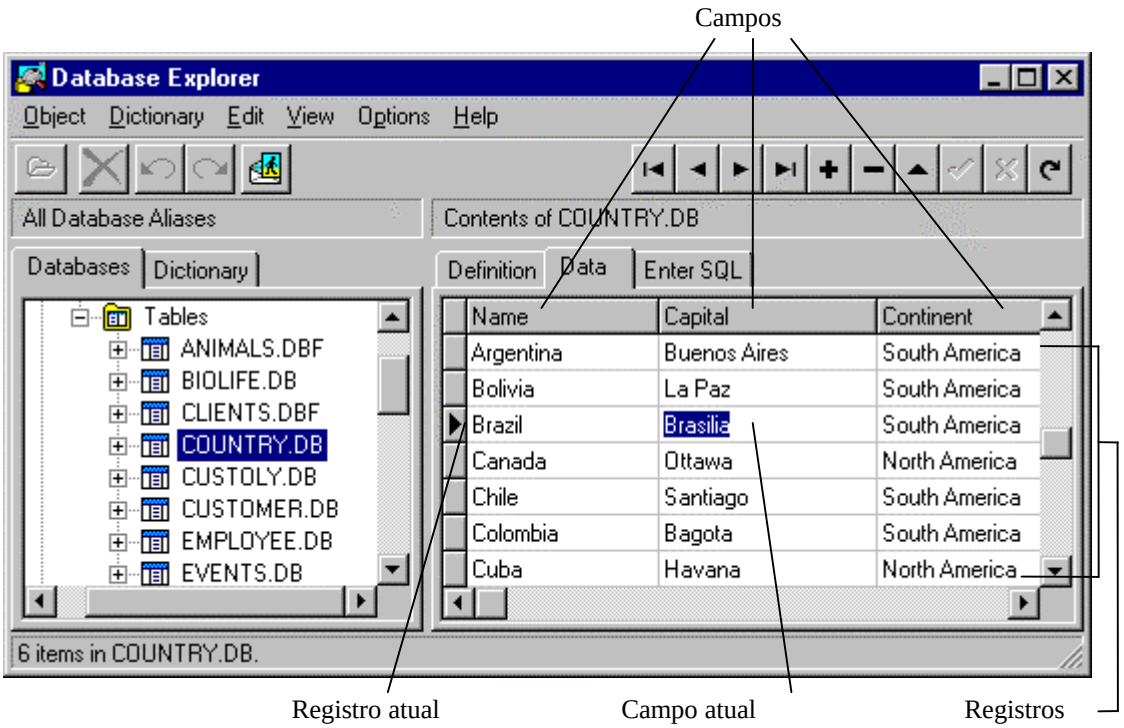
Os aplicativos de banco de dados do Delphi podem ter acesso direto ou indireto às fontes de dados que eles referenciam, isto vai depender de quais componentes você utilizada para acessar o banco de dados. Por exemplo, um apilcativo pode fazer acesso indireto a diversos tipos diferentes de bancos de dados se utilizar componentes do BDE (Borland Database Engine), ADO (ActiveX Data Objects) ou dbExpress, porém se o aplicativo necessitar acessar somente um tipo de banco de dados, como por exemplo Interbase, então recomenda-se fazer o acesso direto utilizando-se componentes IBX (Interbase Express) ou IBObjects entre outros.

O BDE também pode fazer interface com SQL Links da Borland, que permitem acesso a diversos servidores SQL remotos e locais. O servidor local disponível é o InterBase for Windows; entre os servidores remotos, incluem-se o Oracle, o Sybase, o Informix e o InterBase. Se você precisar acessar um banco de dados ou formato de dados diferente, o BDE pode fazer interface com drivers ODBC. Embora o ODBC possa fornecer acesso a fontes de dados, este é o método menos eficiente. Utilize ODBC somente como última opção. Veja na seguinte uma ilustração acerca de como o acesso a bancos de dados funciona no Delphi.



4.1.1. O que é uma Tabela?

Uma tabela em um banco de dados pode ser comparada a um arquivo de registro na linguagem Pascal. Uma tabela tem muitos registros, ou *linhas*, e muitas colunas, uma para cada campo do registro. Você pode ver a estrutura de uma tabela, com seus elementos-chave rotulados, na figura abaixo. Note que na tabela há os conceitos de *registro atual* (o registro com o qual o usuário está trabalhando) e de *campo atual* (o campo ativo do registro atual).



Existem vários aplicativos que você pode utilizar para criar uma tabela de banco de dados, tais como: dBASE, Paradox, Access, entre outros. O Delphi possui uma ferramenta simples para a criação de tabelas, é o Database Desktop. Você também pode utilizar códigos do Delphi para criar uma nova tabela em tempo de execução de um aplicativo.

Para criar tabelas e outras estruturas em bancos de dados relacionais é necessário utilizar outras ferramentas, como por exemplo, IBExpert ou IBConsole, que são usados para manipular bancos de dados Interbase. Também é possível manipular os metadados de bancos de dados relacionais através de código do Delphi.

4.2. Operações com Bancos de Dados

Uma vez tendo uma tabela de banco de dados, você pode efetuar diversas ações, tais como editar valores, inserir novos registros e eliminar registros existentes. Você também pode executar operações em uma janela que tenha uma cópia de alguns dados do banco de dados e, depois, copiar esses dados de volta para o banco de dados. Você verá essas duas etapas diferentes no código de muitos exemplos de aplicativos de banco de dados.

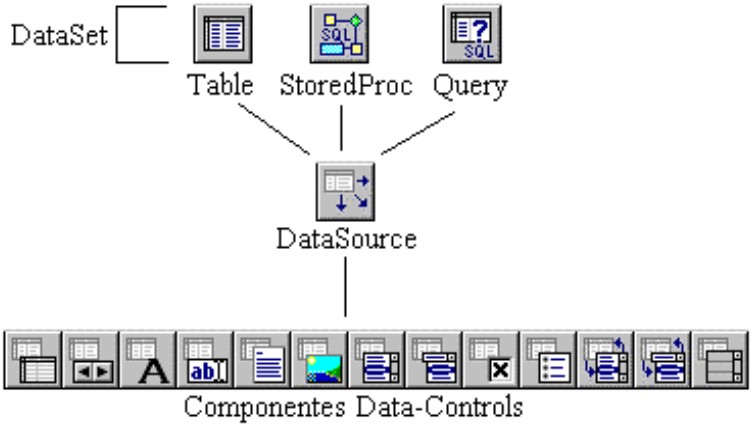
O Delphi oferece uma visão uniforme de acesso a bancos de dados, mas você deve estar ciente do fato de que nem todos os bancos de dados suportam os mesmos recursos.

4.3. Componentes de Bancos de Dados do Delphi

A página da paleta *Data Access* contém componentes usados para interagir com bancos de dados. Todos são componentes não-visuais, já que envolvem conexões, tabelas, queries e elementos similares de bancos de dados.

Na página *Data Control*, há diversos componentes visuais usados para visualizar dados em um formulário. Esses controles são chamados componentes *relativos a dados* (data-aware).

Para acessar um banco de dados no Delphi, geralmente você precisa de uma fonte de dados, descrita pelo componente DataSource. Todavia, o componente DataSource não indica os dados diretamente; ele refere-se a uma tabela ou ao resultado de uma query. Assim, você também precisa de um componente Table ou Query no formulário, conforme pode ver no esquema abaixo.



4.4. Tabelas e Queries

Um objeto Table simplesmente refere-se a uma tabela de banco de dados. Quando você usa um componente Table, precisa indicar o nome do banco de dados que quer usar na propriedade DatabaseName. Você pode informar o próprio nome, um alias ou o caminho do diretório em que estão os arquivos de tabela. O Object Inspector lista os nomes disponíveis, o que depende da instalação do BDE. Você deve indicar o nome do arquivo que contém a tabela na propriedade TableName.

O outro conjunto de dados disponível no Delphi é o componente Query. Normalmente, uma query é mais complexa que uma tabela, porque precisa de uma string de linguagem SQL. Todavia, você pode personalizar uma query usando a SQL de maneira mais fácil do que pode personalizar uma tabela (certamente, desde que você conheça pelo menos os elementos básicos da SQL).

O componente Query também tem uma propriedade DatabaseName, mas não uma propriedade TableName. A tabela está indicada dentro da declaração SQL, armazenada na propriedade SQL.

Você pode usar a instrução SQL em tempo de projeto, abrindo o editor de string SQL ou em tempo de execução. Por exemplo, você pode escrever uma instrução SQL simples como esta:

```
select * from Country
```

em que Country é o nome de uma tabela e o asterisco indica que você quer usar todos os campos da tabela.

A eficiência de uma tabela ou query depende do banco de dados que você estiver utilizando. O componente Table tende a ser mais rápido em tabelas locais, enquanto o componente Query tende a ser mais rápido em servidores SQL, embora esse não seja sempre o caso.

Queries são geralmente utilizadas para unir duas ou mais tabelas e ver o resultado como se fosse uma única tabela armazenada no banco de dados. Enquanto um componente Table refere-se a uma tabela do banco de dados, uma instrução SQL produz uma tabela como resultado. Isso permite que você visualize uma tabela que não esteja no banco de dados, mas que seja o resultado de uma união, uma seleção ou outros cálculos.

O terceiro componente data set é StoredProc, que se refere a procedures locais de um servidor de banco de dados SQL. Você pode executar essas procedures e obter os resultados na forma de uma tabela de banco de dados.

Quando você opera em um data set no Delphi (tal com uma tabela ou query), pode trabalhar em diferentes estados, indicados por uma propriedade State específica, que pode assumir valores diferentes:

- `dsBrowse` indica que o data set está em modo normal de visualização, utilizado para ver e pesquisar os dados.
- `dsEdit` indica que o data set está em modo de edição.
- `dsInsert` indica que um novo registro está sendo adicionado ao data set.
- `dsInactive` é o estado de um data set fechado.
- `dsSetKey` indica que estamos preparando uma pesquisa no data set.
- `dsCalcFields` é o estado de um data set enquanto um cálculo de campo (uma chamada a um manipulador de evento OnCalcField) está sendo preparado.

4.5. Componentes Delphi Relacionados com Dados

Já vimos como é possível conectar uma fonte de dados a um banco de dados, usando uma tabela ou uma query, mas ainda não sabemos como mostrar os dados. Para isso, o Delphi tem muitos componentes que se parecem com os controles normais do Windows, mas que são relacionados com dados. Por exemplo, o componente DBEdit é similar ao componente Edit, e o componente DBCheckBox corresponde ao componente CheckBox. Você encontra todos esses componentes na página Data Controls da paleta Components do Delphi:

- DBGrid é uma grade capaz de exibir toda uma tabela de dados de uma só vez. Ela permite rolagem e navegação e você pode editar seu conteúdo.
- DBNavigator é um conjunto de botões usados para navegar e executar ações no banco de dados.
- DBLabel é usado para exibir o conteúdo de um campo que não pode ser modificado.
- DBEdit é usado para permitir que o usuário veja e modifique um campo.
- DBMemo é usado para permitir que o usuário veja e modifique um grande campo de texto, eventualmente armazenado em um campo memo ou BLOB (que significa, em inglês, Grande Objeto Binário).
- DBImage é usado para mostrar uma figura armazenada em um campo BLOB.
- DBListBox e DBComboBox são usados para permitir que o usuário selecione um único valor de um conjunto especificado. Se este conjunto for extraído de outra tabela do banco de dados ou for o resultado de outra query, você deve utilizar os componentes DBLookupListBox ou DBLookupComboBox.
- DBCheckBox pode ser usado para mostrar e ativar/desativar uma opção, correspondendo à avaliação de uma função.
- DBRadioGroup é usado para fornecer uma série de opções, com um conjunto exclusivo de seleções de botões de rádio, similar aos componentes ListBox e ComboBox.
- DBCtrlGrid é uma grade multi-registro, que pode acomodar diversos outros controles relacionados com dados. Estes controles são duplicados para cada registro do data set.

Todos esses componentes são conectados à fonte de dados usando a propriedade correspondente, DataSource. Muitos deles referem-se a um campo de dados específico da fonte, com a propriedade DataField. Uma vez que você selecione a propriedade DataSource, a propriedade DataField terá uma lista de valores disponíveis na caixa de opções do Object Inspector.

4.6. Os Componentes TField

A importância desse componente não deve ser subestimada. Ainda que freqüentemente ele seja usado nos bastidores, seu papel em aplicativos de bancos de dados é fundamental. Mesmo que você defina objetos específicos desse tipo, a qualquer momento você pode acessar os campos de uma tabela ou query usando suas propriedades Fields, uma matriz de objetos TField. Quando você usa essa abordagem, também quando lida diretamente com um objeto Field, pode usar um conjunto de propriedades "As" para lidar com o campo usando um tipo de dado específico:

```
AsBoolean: Boolean;
AsDateTime: TDateTime;
AsFloat: Double;
AsInteger: LongInt;
AsString: String;
```

Esse código pode ser usado para associar ou ler os valores do campo. Como uma alternativa, você pode utilizar a propriedade Value diretamente. Esta propriedade é definida como um Variant, porque seu tipo depende do tipo de campo.

A maioria das outras propriedade do componente TField, como Alignment, DisplayLabel, DisplayWidth e Visible, reflete elementos da interface com o usuário do campo e é usada com o componente DBGrid. As subclasses de TField são mostradas na tabela 5.1.

Tabela 5.1: As subclasses de TField.

Subclasse	**Definição**
TStringField	Dados de texto de tamanho fixo (pode ter até 255 caracteres, mas depende do tipo de banco de dados)
TIntegerField	Números inteiros na faixa de inteiros long (32 bits)
TSmallIntField	Números inteiros na faixa de inteiros (16 bits)
TWordField	Números inteiros positivos na faixa word ou integers sem sinal (16 bits)
TFloatField	Números reais de ponto flutuante
TCurrencyField	Valores monetários na mesma faixa dos números reais
TBCDField	Números reais com uma quantidade fixa de dígitos depois do ponto decimal.
TBooleanField	Campo com valor booleano
TDateTimeField	Campo com valor de data e hora
TDateField	Campo com valor de data
TTimeField	Campo com valor de hora
TBlobField	Campo com dados aleatórios e sem limite de tamanho
TVarBytesField	Campo com dados aleatórios com até 64 K caracteres
TMemoField	Texto de tamanho arbitrário
TGraphicField	Gráficos de tamanho arbitrário

5. UMA VISÃO DE APLICATIVOS CLIENTE/SERVIDOR

No Delphi você pode trabalhar com dados em tabelas, ou seja, em arquivos normais, como dBASE ou Paradox. Mas se você precisar de um aplicativo mais robusto e seguro, pode considerar a migração de seus dados para um servidor SQL.

O servidor SQL pode residir em um computador servidor conectado a uma rede ou pode estar no mesmo local que a máquina que você está usando para desenvolver seus programas. O Delphi tem uma versão local do servidor SQL da Borland para o MS Windows, o InterBase.

5.1. Acessando um Servidor SQL

O Servidor SQL local pode ser usado como plataforma de destino e como plataforma de desenvolvimento. Quando você o utiliza como plataforma de destino, acaba instalando uma cópia do servidor InterBase, junto com seu programa e com as bibliotecas de que seu programa precisa.

Geralmente, é mais importante utilizar o InterBase somente como plataforma de desenvolvimento. Em vez de desenvolver o aplicativo direto na rede, você pode montá-lo em uma máquina local e então simplesmente alterar o banco de dados de destino no final.

Desenvolver um aplicativo Delphi que acesse um servidor remoto não é muito diferente de desenvolver um aplicativo Delphi que acesse tabelas locais. Entretanto, há algumas diferenças. Por exemplo, geralmente o usuário é solicitado a conectar-se para poder acessar o banco de dados. Outro elemento-chave dos servidores SQL é a presença de um controle de transação mais poderoso. Entre outros recursos incluem-se a execução de stored procedures e a eficiência melhorada no uso de queries em vez de tabelas. Indexação e técnicas de otimização podem mesmo depender de servidores SQL específicos.

Algumas das questões relacionadas com essas diferenças podem ser manipuladas por alguns dos componentes de banco de dados do Delphi. Por exemplo, o componente TDatabase permite que você personalize o procedimento de login ao banco de dados e determine como a transação deve interagir com outras transações simultâneas que acessem as mesmas tabelas (nível de isolamento de transação). O uso desses componentes pode melhorar o controle que você tem sobre o ambiente servidor.

Outro componente específico de servidor é o componente TStoredProc. Ele permite que um programa execute um procedimento presente no banco de dados servidor, eventualmente passando alguns parâmetros e recuperando um valor de retorno. Um terceiro componente útil para manipulação de dados é TBatchMove, que permite a um programa copiar, incluir ou eliminar grupos de registros ou toda uma tabela de dois bancos de dados diferentes.

A única diferença real entre utilizar um banco de dados local e acessar um servidor SQL é que você precisa da versão Client/Server Suite do Delphi, com seus SQL Links, ou utilizar o componentes que fazem acesso direto ao banco de dados. Na realidade, você pode acessar a maioria dos servidores SQL através de ODBC, mas isso está longe de ser a melhor escolha se eficiência for um requisito.

Se optar por utilizar SQL Links para fazer o acesso, então você precisará de diversos passos para instalar e configurar tudo. Você pode instalar o BDE com o Delphi ou separadamente, e ele pode ser configurado usando-se o utilitário BDE Configuration. Com o utilitário de configuração, você pode definir aliases, instalar novos drivers ODBC e definir parâmetros usados para conexão com diversos bancos de dados.

Se você pretende fazer acesso direto ao servidor de dados, então deverá utilizar componente específico. Isto traz grandes vantagens como ganho de eficiência e facilidade para fazer o *deploy* da aplicação, porém sua aplicação ficará dependente do banco de dados escolhido.

5.2. Ferramentas do Servidor InterBase

O Delphi tem aplicativos que podem ser usados para administrar bancos de dados e servidores InterBase locais e remotos. Você pode utilizar também ferramentas de terceiros, como o IBExpert por exemplo.

5.2.1. Server Manager

O Server Manager pode ser utilizado para administrar bancos de dados e servidores InterBase, para definir o modo de inicialização, diretório raiz e propriedades.

5.2.2. IBConsole

O aplicativo IBConsole pode ser usado para configurar bancos de dados Interbase, cadastrar e definir acesso de usuários, efetuar backup e restauração de dados etc. Ele permite ainda executar instruções SQL em um servidor InterBase local ou remoto..

O IBConsole pode ser usado para visualizar o conteúdo de um banco de dados, mas sua finalidade é a configuração e manutenção de um banco de dados. Você pode definir novas tabelas, incluir índices, escrever procedimentos (stored procedures) e assim por diante. Tudo isso é feito usando-se comandos SQL, de modo que, provavelmente, não é tarefa para o usuário comum.

5.2.3. IBExpert

Esta é uma ferramenta muito boa de terceiros que pode ser baixada da internet e utilizada para a manutenção de bancos de dados Interbase/Firebird.

6. UMA INTRODUÇÃO À SQL

Quando você usa um componente TQuery ou TIBQuery em um aplicativo de dados Delphi, você precisa escrever uma instrução SQL na propriedade SQL do componente.

6.1. O que é SQL?

A abreviatura SQL significa Structured Query Language (Linguagem Estruturada de Consulta). É uma linguagem padrão, usada para construir e acessar sistemas de bancos de dados relacionais (RDBMS) de diferentes tipos e em diferentes plataformas de hardware. Apesar de ser uma linguagem padrão, há diferenças entre os dialetos SQL implementados em bancos de dados SQL.

6.2. A Instrução Select

A mais importante instrução SQL é a Select, que é criada em cima de três cláusulas:

- A cláusula `select` indica lista de campos, que você quer ver no resultado da query; usando o símbolo de asterisco (*) em vez de uma lista de campos, você pode selecionar todos os campos.
- A cláusula `from` indica as tabelas que você quer considerar na montagem da query.
- A cláusula `where` indica algum critério de seleção. Se não houver a cláusula `where`, todos os campos são selecionados.

Por exemplo, se você quiser selecionar todas as linhas (ou registros) da tabela Country.DB, pode escrever: `select * from country`

Para especificar as linhas (ou registros) que deseja ver no resultado, você pode escrever: `select Name, Capital from Country`

6.3. A cláusula Where

Se você quiser somente algumas e não todas as linhas, você pode incluir a cláusula `where`:

```
select Name, Capital from Country
where Population > 20000000
```

Isso retorna somente os países que tenham mais de 20 milhões de habitantes. Na cláusula `where`, você pode escrever diversas expressões diferentes, inclusive uma expressão para pesquisar um único elemento, como em:

```
select * from country
where Name = 'Argentina'
```

Certamente, você também pode escrever condições múltiplas ao mesmo tempo. Você pode mesclar duas condições com `and` e indicar que elas devem ser atendidas ao mesmo tempo, `ou` pode usar or para indicar que um registro deve atender a uma das condições:

```
select * from country
where Continent = "South America" and Population < 10000000

select Name, Capital from country
where Name <= "Brasil" or Capital >= "Ottawa"
```

A primeira instrução seleciona os países da América do Sul com menos de 10 milhões de habitantes. A segunda instrução seleciona os países que tenham um nome que precedam o do *Brasil* na ordem alfabética e os que tenham capital com um nome depois de *Ottawa*.

6.4. Evitando Repetições

Para evitar elementos repetidos no resultado, você pode incluir mais uma palavra-chave SQL, `distinct`, que força a remoção das repetições.

```
Select distinct Continent from Country
```

6.5. Fazendo uma Junção

Na maioria dos casos, as instruções SQL referem-se a duas ou mais tabelas. Quando você trabalha com duas tabelas, pode juntá-las ou usar o resultado e uma tabela para expressar a condição usada na segunda.

Quando você trabalha com duas ou mais tabelas-fonte, geralmente você as junta. Em SQL não há instruções ou cláusulas específicas para expressar uma junção (apesar de isto ser possível no dialeto SQL de alguns RDBMS, incluindo o InterBase).

Simplesmente, você pode trabalhar com duas tabelas e juntá-las de maneira apropriada usando a cláusula where para comparar os valores de dois campos. Por exemplo, podemos juntar as tabelas Pedidos e Clientes para achar a data de cada pedido e a companhia envolvida, escrevendo:

```
select Orders.OrderNo, Orders.SaleDate,
  Customer.Company, Customer.Contact
from Orders, Customer
where Orders.CustNo = Customer.CustNo
```

Exatamente da mesma maneira, você pode juntar três tabelas usando duas condições. Neste caso, também queremos saber o nome do empregado que processou o pedido:

```
select orders.OrderNo, Orders.saleDate,
  Customer.Company, Customer.Contact,
  Employee.LastName
from Orders, Customer, Employee
where Orders.CustNo = Customer.CustNo and
  Orders.EmpNo = Employee.EmpNo
```

Com a seguinte instrução SQL, recuperamos o valor pago por cada cliente que fez algum pedido através do empregado *Johnson*:

```
select Orders.AmountPaid, Customer.Company
  from Orders, Customer, Employee
where Orders.CustNo = Customer.CustNo
  and Employee.LastName = "Johnson"
```

6.6. Escolhendo uma ordem

Outra cláusula SQL é `order by` (duas palavras separadas), que determina a ordem dos valores na query resultante:

```
select * from country where Population > 10000000
  order by Continent, Name
```

Essa declaração retorna uma tabela de países ordenada por continente e, então, por nome.

6.7. Calculando Valores

Na cláusula select em vez de um campo de uma tabela, você pode ter o resultado de um cálculo. Por exemplo, você pode calcular o número de empregados com esta instrução:

select Count (*) from employee

Para ver um exemplo mais complexo, você pode calcular a quantidade de pedidos tirados por um empregado, o valor total e a média:

```
select Sum (Orders.AmountPaid),
  Count (Orders.AmountPaid), Avg (Orders.AmountPaid)
  from Orders, Employee
where Orders.EmpNo = Employee.EmpNo and
  Employee.LastName = "Johnson"
```

6.8. Definindo Grupos

Além de fazer cálculos com o resultado e uma query, você pode calcular um valor para cada grupo de elementos, conforme indicado pela cláusula `goup by`:

```
select Employee.LastName, Sum(Orders.AmountPaid)
  from Orders.Employee
where Orders.EmpNo = Employee.EmpNo
group by Employee.LastName
```

O resultado é uma lista de empregados, cada um deles com o total de pedidos tirados.

7. REFERÊNCIAS BIBLIOGRÁFICAS

CANTU, Marco. *Dominando o Delphi – A Bíblia*. Makron Books.

BLUE, Ted KASTER, John LIEF, Greg SCOTT, Loren. *Desenvolvendo Bancos de Dados em Delphi*. Makron Books.

RIBEIRO, Sebastião Elivaldo. *Criando Componentes no Delphi*. Visual Books.

Borland: http://www.borland.com

Torry´s Delphi Pages: http://homepages.borland.com/torry

Project Jedi: http://www.delphi-jedi.org/