Série Visual Studio 2005
Express Editions

C#

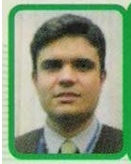
Desenvolvendo uma Aplicação Comercial

Escrito por três grandes evangelistas, Alexandre Tarifa (MVP Visual Developer Visual Basic .NET), Emerson Facunte (MVP Visual Developer ASP/ASP.NET) e Marcus Garcia (MVP Visual Developer - Visual Basic .NET), o livro traz uma proposta inovadora, onde o leitor aprenderá de forma prática e objetiva os poderosos recursos da ferramenta C# 2005 Express Edition.

Na parte final do livro, o leitor irá desenvolver uma aplicação comercial, aprendendo desde a parte de negócios, passando pelo diagrama de classes e o desenvolvimento da aplicação, até a distribuição da mesma.

Principais tópicos abordados:

- Fundamentos do Framework .Net
- ADO.NET
- Conhecendo os recursos da IDE
- Fundamentos da linguagem C#
- Declaração de variáveis
- Laços
- Estrutura condicional
- Classes
- Diagrama de classes
- SQL fundamental
- Criação do banco de dados
- Criação das classes
- Desenvolvimento da aplicação passo a passo
- Distribuição da aplicação



Alexandre Tarifa
É Most Valuable Professional - Visual Developer - Visual Basic, MCAD (Microsoft Certified Application Developer) e MCT (Microsoft Certified Trainer). Bacharel pela UMESP e pós-graduado pela Universidade Federal de São Carlos em Ciência da Computação. Analista de Sistemas e Líder do grupo de usuários Codificando.net São Paulo (sp.codificando.net).

Emerson Facunte
É Most Valuable Professional - Visual Developer - ASP/ASP.NET, MCP 70-300 (Analyzing Requirements and Defining Microsoft .NET Solution Architectures) e co-owner Framework .NET Microsoft Brasil. Publicou 7 livros, também editados pela Brasport, e mais de 100 artigos e ministrou palestra para cerca de 15.000 pessoas em todo o país. É líder do grupo Go.Net, consultor sênior do grupo DevMedia e Arquiteto de Software do Grupo Saraiva.

Marcus Garcia
MVP Visual Developer - Visual Basic .NET, Diretor da INETA BRASIL (International .NET Association), Gerente de Tecnologia - Freire Advogados & Associados, Desenvolvedor e Administrador de redes, Bacharel em Engenharia Elétrica e Ciência da Computação, Colunista e articulista em diversos sites e revistas de grande expressão nacional e internacional, palestrante Microsoft e palestrante no TECH ED 2005 - Maior evento Microsoft do ano.

Visite nosso site
www.brasport.com.br

ISBN 85-7452-271-6

Série Visual Studio 2005
Express Editions

C#

Desenvolvendo uma Aplicação Comercial

Prefácio por
Leonardo Tolomelli
Rogério Panigassi

Alexandre Tarifa • Facunte • Marcus Garcia

C#

Desenvolvendo uma Aplicação Comercial

volume 2

Alexandre Tarifa
Emerson Facunte
Marcus Garcia

Sumário

Parte I
Desenvolvimento .NET

1. Introdução **3**
Apresentação 3
Quem deve ler este livro? 3
Versões Express 4
Visual Studio 2005 – Novo Cenário 4
Posicionamento das ferramentas Express 5
Novos recursos 8

2. Framework .NET **10**
Common Language Runtime 11
Base Class Library 12
ADO.NET 12

3. Instalando o C# Express Edition **17**

4. C# Express Edition – Primeiros Passos **31**

Parte II
Essencial C#

1. Variáveis **39**
Tipos simples de valor em C# 40
Enumerações 43
Funções para tratamento de Strings 45

2. Estrutura Condicional **47**
Tomada de Decisão 47
Tomada de Decisão do Tipo Escolha 50

3. Laços **52**
Laço For 52
Laço foreach/in 53

Laço While 54
Laço Do/While 54

4. Tratamento de Erros – Exception 56
Visão Geral 56
Estrutura 56
Gerando suas próprias exceções 57
Exemplos 58

5. Classes 63
Visão geral 63
Estrutura da classe em C# 63

Parte III
Desenvolvimento da Aplicação

1. Negócio 67
Visão Geral do Projeto 67
Banco de Dados 67
Diagrama de classes 70

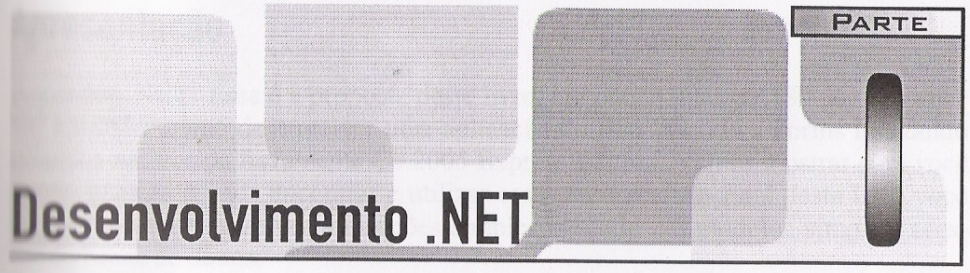
2. Criando a Aplicação 73
Criação do projeto 73
Interface gráfica da aplicação 74
Integrando Interface e Regra de Negócio 130

3. Distribuição 159
Algumas configurações 159
Prática 163

Apêndice. Introdução SQL 167
SQL 167
Principais comandos 167

Referências Bibliográficas 173

Índice Remissivo 175



Microsoft .net

CAPÍTULO 1

Introdução

Apresentação

Aprender .NET! Essa é a proposta deste livro, em pouco mais de 180 páginas você vai aprender como desenvolver uma aplicação do tipo Windows Forms utilizando recursos nativos da ferramenta C# 2005 Express Edition. Vamos mostrar para você como, quando e onde encontrar e utilizar esses recursos. Ao final deste livro você estará apto a desenvolver aplicações simples baseado em modelos simples, acessando bases de dados.

Se você é um estudante, um entusiasta ou aprendiz adquira a base necessária para mergulhar no mundo da tecnologia .NET, que está ganhando diariamente novos usuários tendo em vista a facilidade de utilização e o tempo curto para aprendizado.

Aplicações do tipo Windows Forms

É um conjunto de classes do Framework .NET para o desenvolvimento de aplicações baseadas em *Windows* (*client*) e *Smart Clients*.

Aplicações do tipo Web Forms

São aplicações no modelo Web baseadas em ASP.NET para geração dinâmica de páginas. Assim como as aplicações Windows Forms, esse modelo é baseado num conjunto de classes do Framework .NET.

Quem deve ler este livro?

Todo e qualquer profissional de informática que queira ingressar no mundo do desenvolvimento .NET deve ler este livro. O foco principal é ensinar os recursos básicos do desenvolvimento em .NET para aplicações do tipo Windows Forms, portanto, trata-se de um livro desenvolvido para iniciantes, de forma simples, direta e focada em resultados imediatos.

Se você já é um desenvolvedor de outras plataformas, entenda como funciona o Framework 2.0. Se está querendo migrar para .NET poderá analisar, praticar e tirar suas próprias conclusões sobre a plataforma.

Você encontrará de forma resumida em cada capítulo deste livro, fundamentos, métodos e técnicas de desenvolvimento necessários no seu dia-a-dia.

O QUE PRECISO PARA EXECUTAR OS EXEMPLOS DESTE LIVRO?

Para executar os exemplos deste livro será necessário um sistema operacional Windows 2000 ou superior e o C# Express.

Versões Express

Muito se tem falado sobre as versões Express 2005. Ainda assim há muita gente que tem dúvidas quanto às vantagens, limitações e recursos. Podemos afirmar com certeza que são, sem sombra de dúvidas, muito produtivas e de fácil operação, assim como a versão Profissional do Visual Studio 2005. Os novos recursos são incríveis. Componentes inovadores fazem com que sua experiência dentro da ferramenta seja muito mais agradável e produtiva. Ficou muito mais fácil distribuir aplicações Windows Forms com a chegada do Click Once (tecnologia de distribuição de aplicações pela rede Internet, Intranet e servidores de arquivos), que realmente cumpre o que promete, nesse caso estamos falando de aplicações Windows sendo distribuídas pela web de maneira simples e muito funcional. Para a turma do SQL Server, temos agora o SQL Server Express, versão que muitas pessoas falam ser o novo MSDE. Creio que está mais para um SQL Server Lite, pois possui até IDE própria, recurso que não existe no MSDE.

Visual Studio 2005 – Novo Cenário

A Microsoft dividiu suas ferramentas de desenvolvimento para atender a todo o público. De aprendizes até testers, todos podem aproveitar a flexibilidade e a eficiência do mundo .NET.

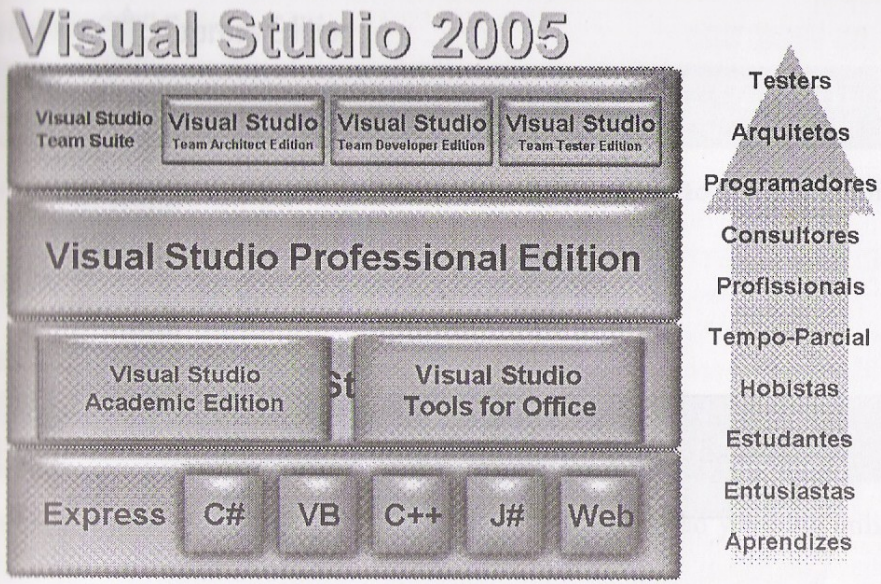


Figura 1.1 – Diagrama do Visual Studio 2005

A versão Standard é mais indicada para profissionais que não dedicam seu tempo integral ao desenvolvimento de sistemas, por isso é indicada para *hobistas* e profissionais que têm outra atividade como principal além de desenvolver sistemas. Essa versão também é indicada para quem quer desenvolver integração com o Office.

A versão Professional Edition é a mais indicada para todos os profissionais de desenvolvimento de sistemas e consultores, pois trata-se da suite completa do produto.

A versão completa é a Team Suite, que contém as versões Team Architect, Team Developer e Team Tester no mesmo pacote.

Posicionamento das ferramentas Express

Na base do diagrama encontramos as ferramentas Express cujo público principal são os estudantes, entusiastas e aprendizes. Com essa versão a Microsoft pretende difundir a estratégia .NET para todas as universidades do Brasil. Já existe, é claro, um serviço/produto Acadêmico nesse sentido (AA – Academic Alliance), porém, as ferramentas Express levam até você muitas facilidades encontradas na suite Profissional do Visual Studio 2005, por custo muito reduzido.

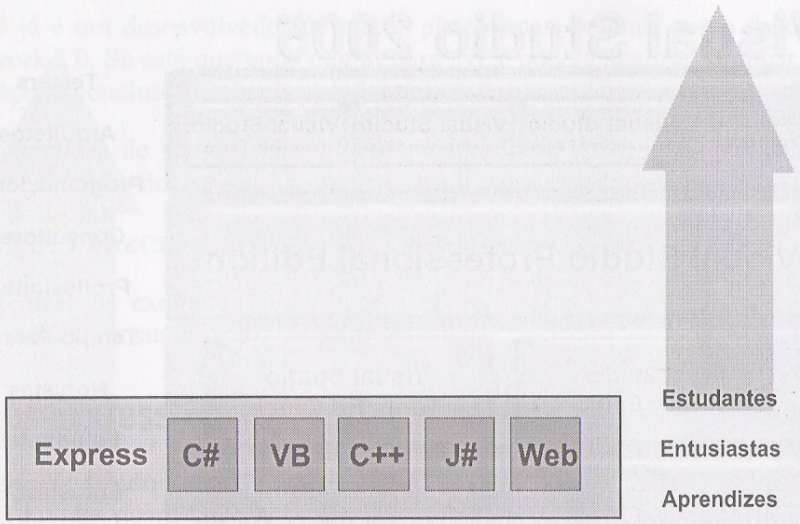


Figura 1.2 – Diagrama das Versões Express

Os produtos Express são compostos pelas seguintes ferramentas:

Visual Web Developer 2005 Express Edition

Ferramenta para construção de sites dinâmicos (Asp.NET) e Web-Services utilizando todas as linguagens Visual Basic .NET, C# e J#.

Para instalar:http://go.microsoft.com/?linkid=2676894

Visual Basic 2005 Express Edition

Ferramenta para aprendizagem de fundamentos de programação Windows utilizando Visual Basic .NET.

Para instalar: http://go.microsoft.com/?linkid=2676890

Visual C# 2005 Express Edition

Ferramenta para aprendizagem de fundamentos de programação Windows utilizando Visual C#.

Para instalar: http://go.microsoft.com/?linkid=2676890

Visual C++ 2005 Express Edition

Ferramenta para aprendizagem de fundamentos de programação Windows utilizando o C++.

Para instalar: http://go.microsoft.com/?linkid=2676890

Visual J# 2005 Express Edition

Ferramenta para aprendizagem de fundamentos de programação Windows utilizando J#.

SQL Server 2005 Express Edition

Ferramenta de banco de dados baseada no SQL Server 2005.

Para instalar:

http://www.microsoft.com/downloads/details.aspx?FamilyId=97383299-E74B-4AF8-90C7-FA0CA20A10C4&displaylang=en

Novos recursos

As ferramentas ganharam alguns recursos interessantes, dos quais podemos destacar:

- ☑ Novos Starter Kits – modelos prontos de aplicações, que facilitam o entendimento e diminuem a curva de aprendizado das ferramentas.
- ☑ Novo Start Page – página inicial da ferramenta que disponibiliza informações on-line, agrupadas por diversas categorias, além de funções operacionais dos projetos.
- ☑ Code-Snippet – fragmentos de códigos já programados para serem utilizados em sua aplicação.

Compatibilidade com outras versões

As ferramentas Express são totalmente compatíveis com as versões Standard e Professional do Visual Studio 2005, portanto, faça sem medo de ter que fazer novamente.

Conclusão

As ferramentas Express chegaram para mostrar uma nova forma de você aprender .NET com baixo impacto e investimento reduzido. Aprendendo com ferramentas Express você estará apto a trabalhar com qualquer ferramenta da família Microsoft Visual Studio.

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

Microsoft .net Framework

CAPÍTULO 2

Framework .NET

O Framework .NET é um ambiente totalmente voltado para o desenvolvimento. Cria um ambiente de execução (runtime) com o objetivo de gerenciar e executar aplicações. A Microsoft criou o Framework .NET para tornar real e viável a interconexão de qualquer tipo de aplicação, dispositivo ou sistemas.

Seus principais objetivos são:

- ☑ Não ser difícil
- ☑ Extensibilidade
- ☑ Práticas e padrões baseados na Web
- ☑ Padronização de modelos de programação

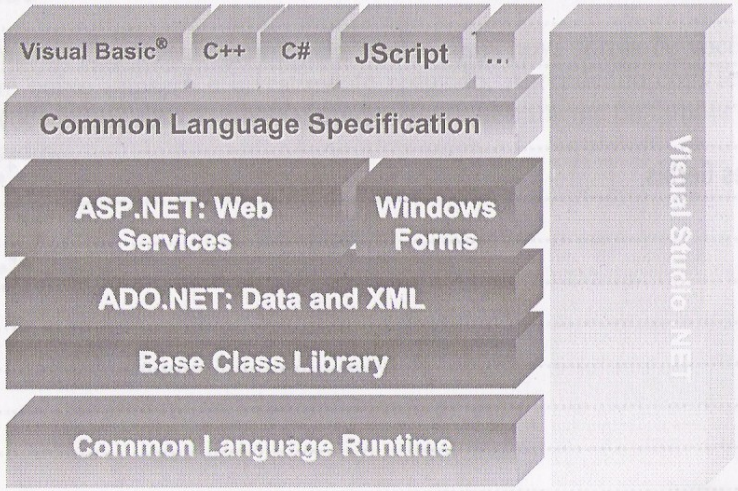


Figura 2.1 – Diagrama do Framework .NET

Na figura 2.1, temos o diagrama do Framework .NET, que é composto por várias camadas, as quais vamos detalhar a seguir:

Common Language Runtime

A CLR é o ambiente responsável por gerenciar a execução de todo o código, portanto, todo código escrito sob a tecnologia .NET é compilado automaticamente pela MSIL (Microsoft Intermediate Language). Ao ser compilada pela MSIL, sua aplicação recebe o metadados (figura 2.2), que são as informações que descrevem toda a sua aplicação e o que pode ou não pode ser feito por ela.

Seus principais objetivos são:

- ☑ Simplificar o desenvolvimento
- ☑ Tornar a execução robusta e segura
- ☑ Suporte a múltiplas linguagens de programação
- ☑ Distribuição e o gerenciamento simplificado

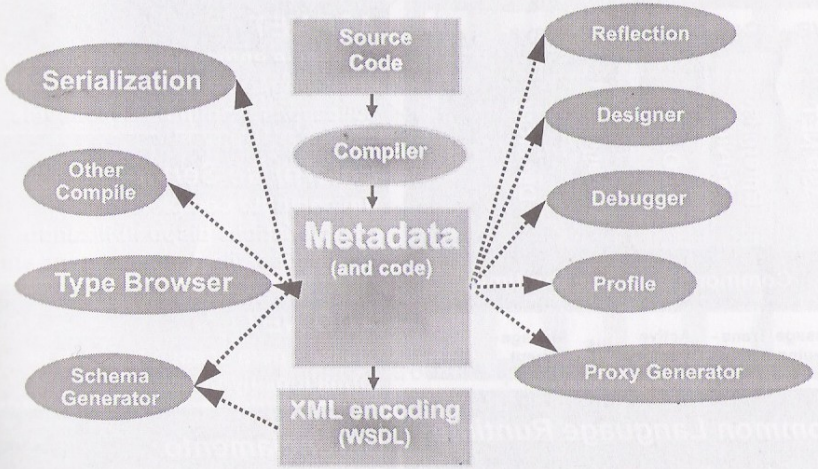


Figura 2.2 – Metadados

A CLR precisa descobrir como executar o aplicativo, para isso utiliza o metadados. Toda CLR é composta por código gerenciado, ou seja, administra a execução de sua aplicação fornecendo benefícios e recursos quando necessário. Alguns benefícios e recursos podemos citar aqui como sendo fundamentais ao funcionamento de toda a sua aplicação: integração entre linguagens, controle de versões e erros etc.

Algumas características do metadados:

- ☑ Base para o modelo de programação simplificado

- ☑ Geração automática
- ☑ Embutido dentro de uma .DLL ou .EXE
- ☑ Formato binário
- ☑ Conversão para/de XML Schema e de/para COM type library

Base Class Library

O Class Library é a segunda camada do Framework .NET. Nessa coleção de classes você pode encontrar várias funcionalidades que dão suporte e facilitam o desenvolvimento. Essa "base de classes" é suficiente para cobrir uma grande variedade de cenários e situações no desenvolvimento de aplicações e componentes. Nessa extensa lista de classe podem-se encontrar tipos para atender a várias áreas e tipos de sua aplicação (figura 2.3).

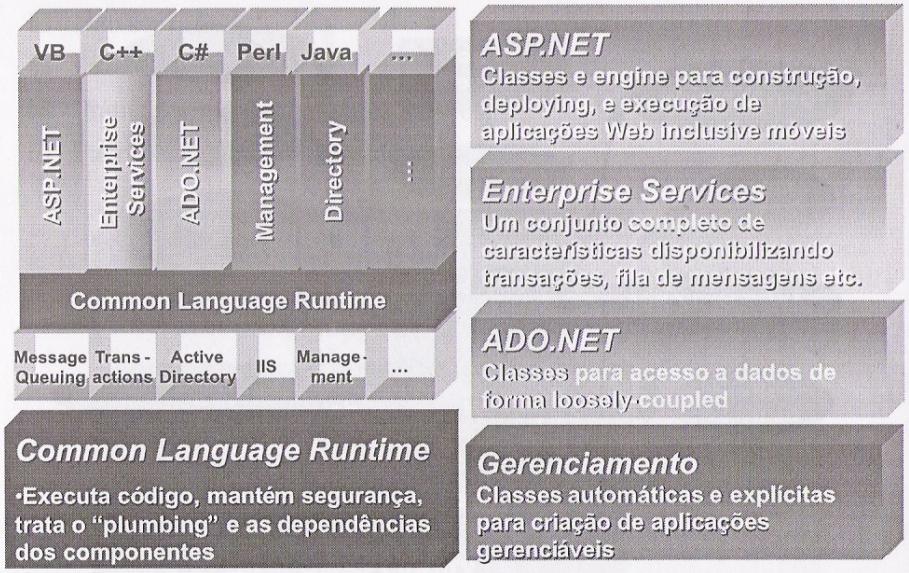


Figura 2.3 – Class Library

ADO.NET

O ADO.NET é muito mais que uma classe de conexão a banco de dados, viabilizando a utilização de diversas classes para operação com dados. É a camada de conexão a banco de dados e está completamente integrada à tecnologia .NET. Possui classes nativas de acesso direto a bancos de dados, chamados de Providers, como o **System.data.SqlClient** – para acesso ao SQL Server – e o **System.data.OracleClient** – para acesso ao banco de dados Oracle –, e também através das classes **System.data.OleDb** – para acesso a qualquer outro banco de dados – e **System.data.odbc** – para acesso através de drivers específicos.

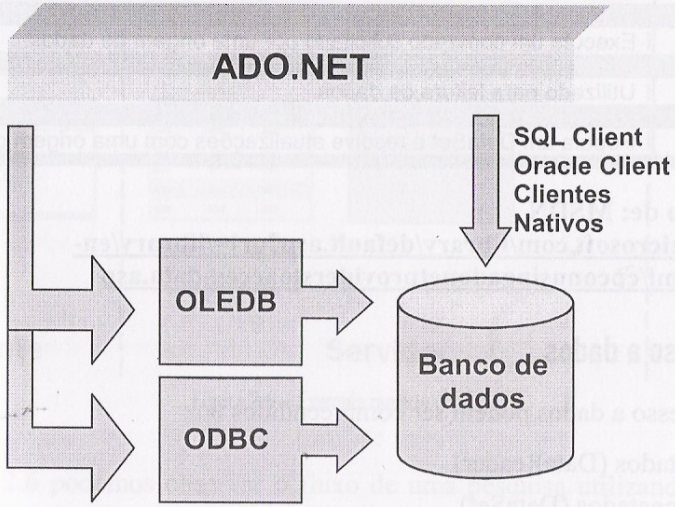


Figura 2.4 – Diagrama ADO.NET.

Usando Provedores para acesso a dados

Um Provedor de dados é uma ponte entre uma aplicação e sua origem de dados, utilizado para recuperar dados de uma origem e executar atualizações, inclusões e exclusões.

Em seguida, temos uma lista de Provedores de dados do Framework .NET.

Provedores de dados	Descrição
Provedor para SQL Server	Para Microsoft® SQL Server™ Versão 7.0 ou Superior.
Provedor para OLE DB	Para origens de dados compatíveis com OLE DB.
Provedor para ODBC	Para origens de dados usando ODBC.
Provedor para Oracle	Para origens de dados ORACLE.

O **Connection**, **Command**, **DataReader** e **DataAdapter** são objetos que representam os elementos dos Provedores de dados. A tabela abaixo descreve esses objetos.

Objeto	Descrição
Connection	Estabelece a conexão com uma origem de dados específica.
Command	Executa um comando solicitado por uma origem de dados.
DataReader	Utilizado para leitura de dados.
DataAdapter	Popula um DataSet e resolve atualizações com uma origem de dados.

Texto extraído de: MSDN
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconusingadonetproviderstoaccessdata.asp

Tipos de acesso a dados

Os tipos de acesso a dados podem ser compreendidos em:

- ☑ Conectados (DataReader)
- ☑ Desconectados (DataSet)

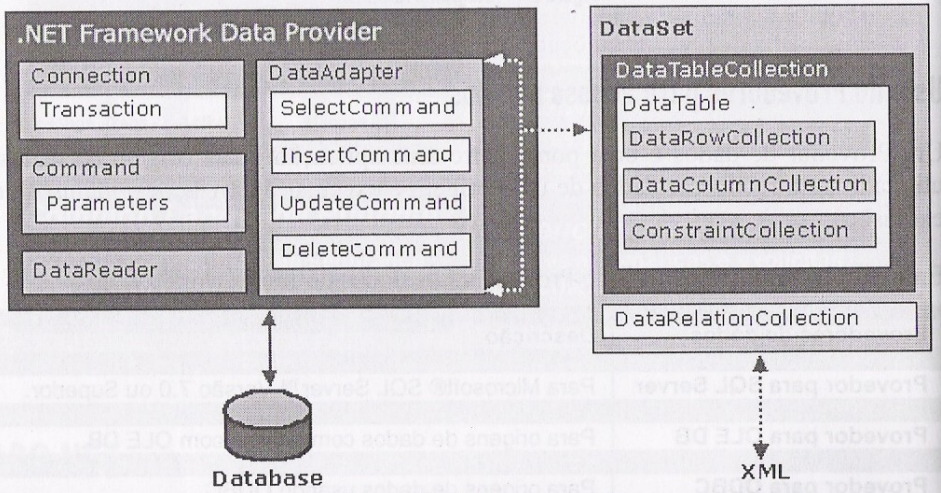


Figura 2.5 – Acesso a dados.

DataReader (Conectado)

O DataReader é utilizado para recuperar informações do Banco de Dados de forma otimizada. Após a leitura das informações, é disponibilizado um mecanismo de manipulação dos dados.

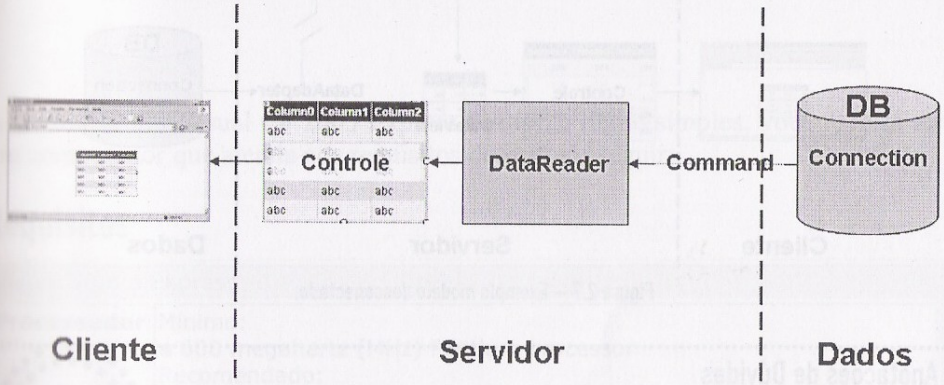


Figura 2.6 – Exemplo modelo conectado.

Na figura 2.6 podemos observar o fluxo de uma pesquisa utilizando um modelo conectado.

DataSet (Desconectado)

O DataSet é totalmente baseado num modelo de tabelas que armazenam informações em memória, criando, dessa forma, total independência do banco de dados, não precisando manter-se conectado o tempo todo.

O DataSet é utilizado para recuperar informações do Banco de Dados de forma estruturada, organizando os dados em tabelas, colunas e linhas.

Na figura 2.7 podemos observar o fluxo de uma pesquisa utilizando um modelo desconectado.

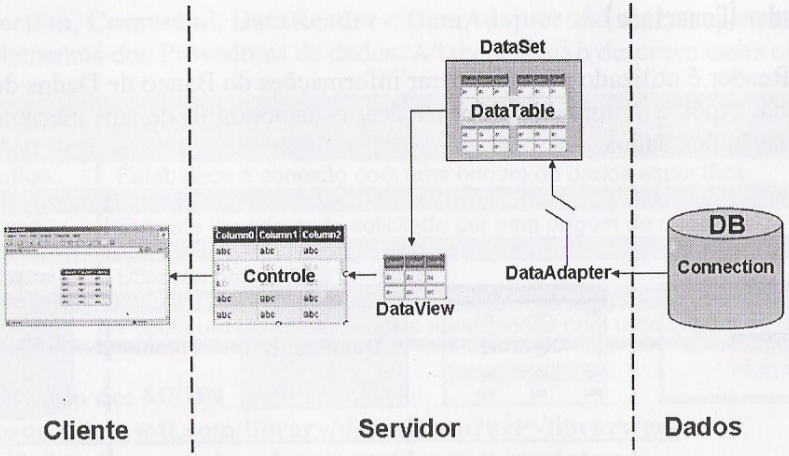


Figura 2.7 – Exemplo modelo desconectado.

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

CAPÍTULO 3

Instalando o C# Express Edition

A instalação do Visual C# 2005 Express Edition é muito simples, você precisa ter um computador que atenda aos requisitos descritos a seguir:

Requisitos

Visual Studio Express	
Processador	Mínimo: ■ 600 megahertz (MHz) Pentium processor Recomendado: ■ 1 gigahertz (GHz) Pentium processor
Sistema Operacional	■ Microsoft® Windows® 2000 Professional SP4 ■ Microsoft® Windows® 2000 Server SP4 ■ Microsoft® Windows® 2000 Advanced Server SP4 ■ Microsoft® Windows® 2000 Datacenter Server SP4 ■ Microsoft® Windows® XP Professional x64 Edition (WOW) ■ Microsoft® Windows® XP Professional SP2 ■ Microsoft® Windows® XP Home Edition SP2 ■ Microsoft® Windows® XP Media Center Edition 2002 SP2 ■ Microsoft® Windows® XP Media Center Edition 2004 SP2 ■ Microsoft® Windows® XP Media Center Edition 2005 ■ Microsoft® Windows® XP Tablet PC Edition SP2 ■ Microsoft® Windows Server ™ 2003, Standard Edition SP1 ■ Microsoft® Windows Server ™ 2003, Enterprise Edition SP1 ■ Microsoft® Windows Server ™ 2003, Datacenter Edition SP1 ■ Microsoft® Windows Server ™ 2003, Web Edition SP1 ■ Microsoft® Windows Server ™ 2003, Standard x64 Edition SP1 (WOW) ■ Microsoft® Windows Server ™ 2003, Enterprise x64 Edition SP1 (WOW) ■ Microsoft® Windows Server ™ 2003, Datacenter x64 Edition SP1 (WOW) ■ Microsoft® Windows Server ™ 2003 R2, Standard Edition ■ Microsoft® Windows Server ™ 2003 R2, Standard x64 Edition (WOW) ■ Microsoft® Windows Server ™ 2003 R2, Enterprise Edition ■ Microsoft® Windows Server ™ 2003 R2, Enterprise x64 Edition (WOW)

Visual Studio Express	
	■ Microsoft® Windows Server ™ 2003 R2, Datacenter Edition ■ Microsoft® Windows Server ™ 2003 R2, Datacenter x64 Edition (WOW) ■ Microsoft® Windows Vista ™ A instalação do Visual Studio 2005 em Intel Itanium (IA64) não é suportada.
RAM	Mínimo: ■ 192 megabytes (MB) Recomendado: ■ 256 MB
Disco Rígido	1.3 GB de espaço mínimo
Drive de CD ou DVD	Não é necessário (pela Internet)
Vídeo	Mínimo: ■ 800 x 600 256 cores Recomendado: ■ 1024 x 768 High Color — 16-bit
Mouse	Microsoft mouse ou compatível

É claro que é preciso também possuir o Framework 2.0; entretanto, o mesmo é instalado automaticamente durante a instalação do Visual C# 2005 Express Edition. Você vai ver a seguir toda a instalação passo a passo para ilustrar totalmente essa etapa.

Vá para a Internet e acesse o site:

http://msdn.microsoft.com/vstudio/express/visualcsharp/, onde pode baixar o software de instalação. Neste site você pode também encontrar dicas de funcionamento e recursos.

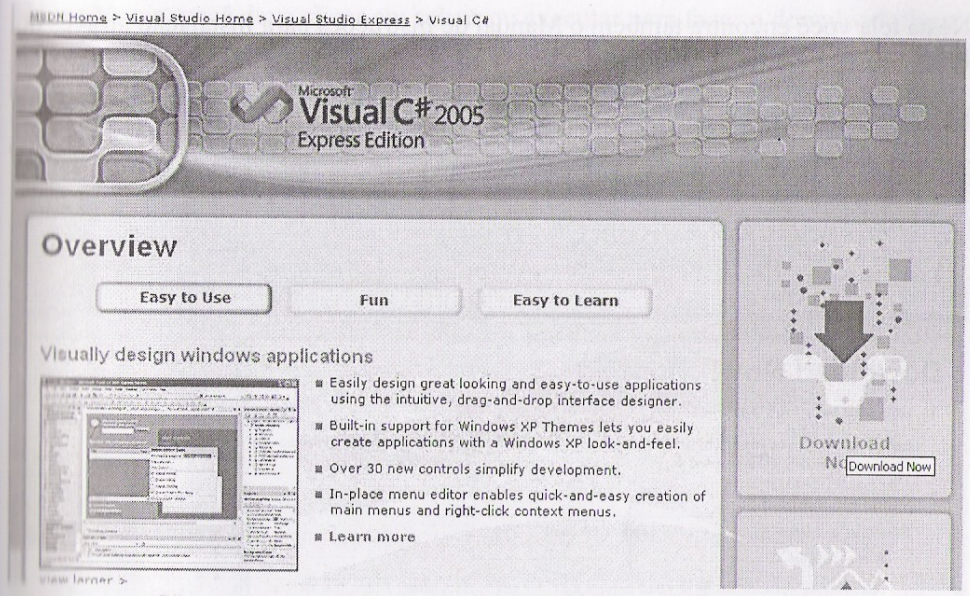


Figura 3.1 – Site do Produto.

Clique na opção Download (figura 3.1) ao lado direito do site e você vai encontrar informações de como instalar, desinstalar e registrar sua cópia após a instalação (figura 3.2).

Nesta tela você encontra também o Manual de Instruções para Instalação através do link:

http://msdn.microsoft.com/vstudio/express/support/install/

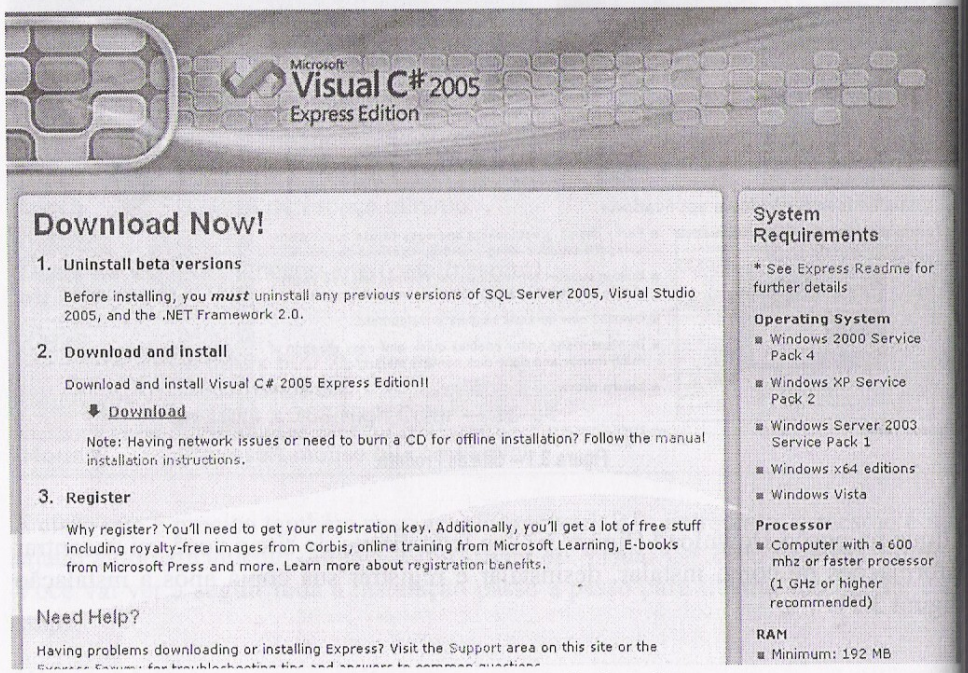


Figura 3.2 – Tela de Download do site do produto.

Na seqüência, você deve clicar em salvar ou executar para fazer o download (figura 3.3).

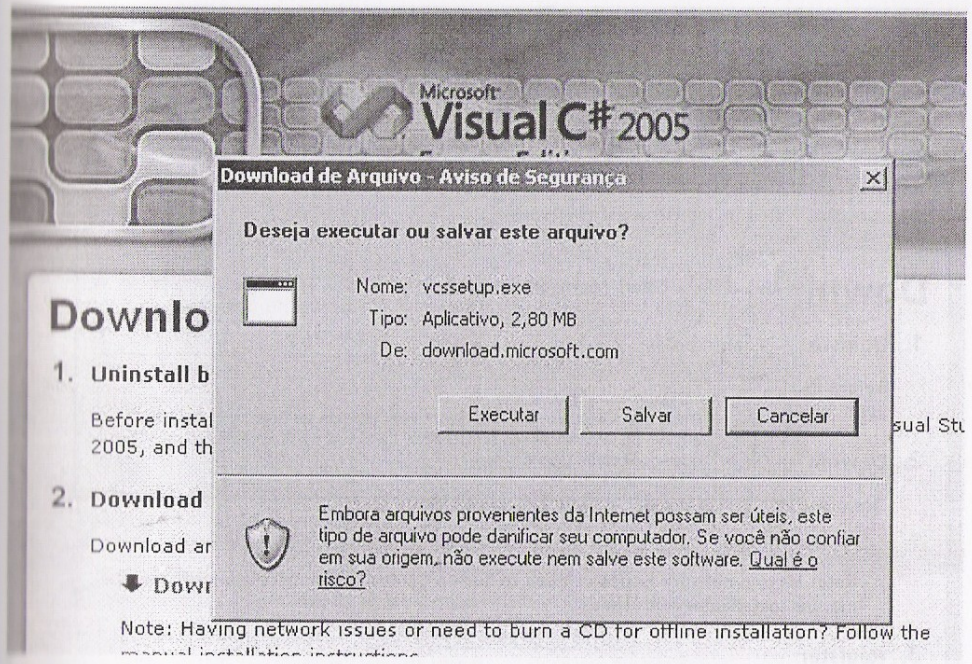


Figura 3.3 – Download do pacote de instação.

Caso esteja utilizando o Windows XP será exibido o aviso de segurança; apenas clique em Executar e prossiga (figura 3.4).

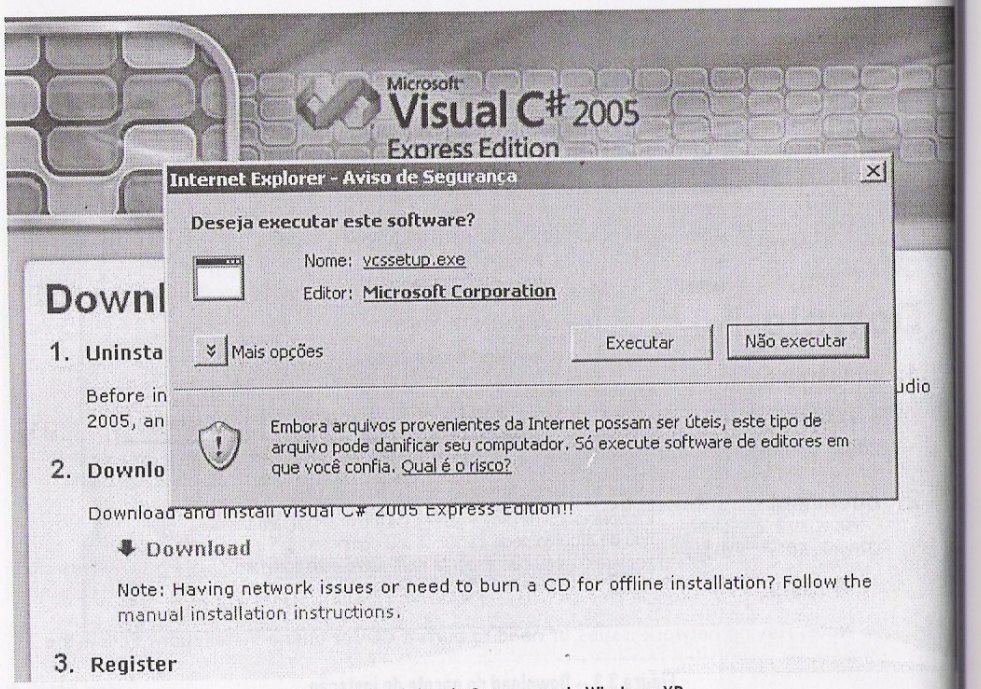


Figura 3.4 – Aviso de Segurança do Windows XP.

Após o download do pacote de instalação, aparecerá a tela de Bem Vindo ao Setup do Visual C# 2005 Express Edition (figura 3.5). Clique em Next.

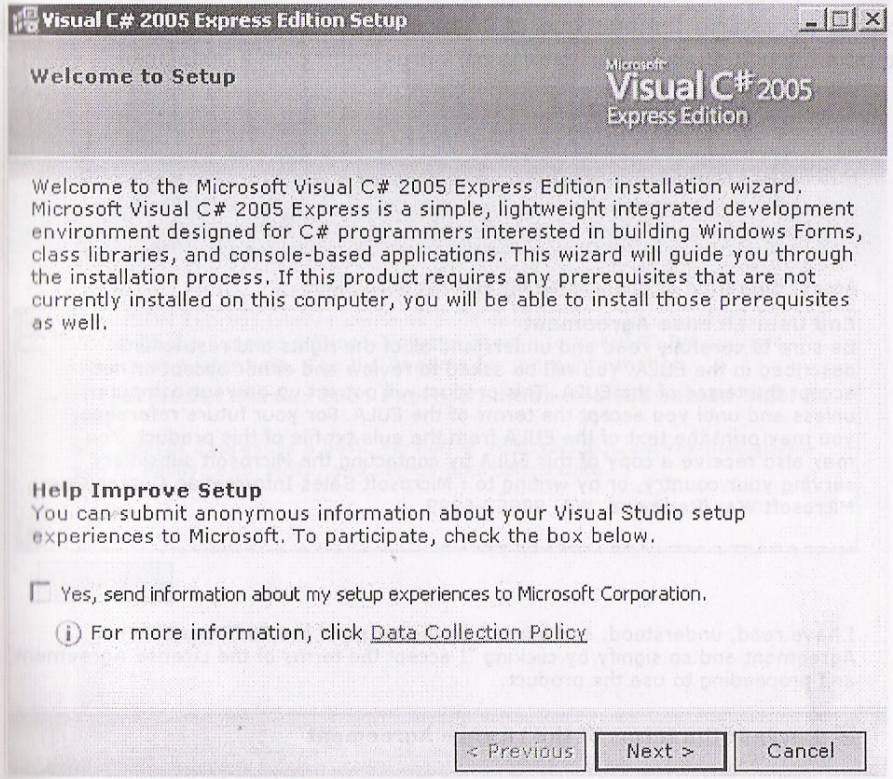


Figura 3.5 – Tela de Boas Vindas do Visual C# 2005 Express Edition.

Na seqüência você deverá clicar no CheckBox "I Accept the terms of the License Agreement" (Figura 3.6).

Nota: É necessário lembrar que as Versões Express são todas gratuitas, porém, aceitar a licença de uso é obrigatório para prosseguir com a instalação.

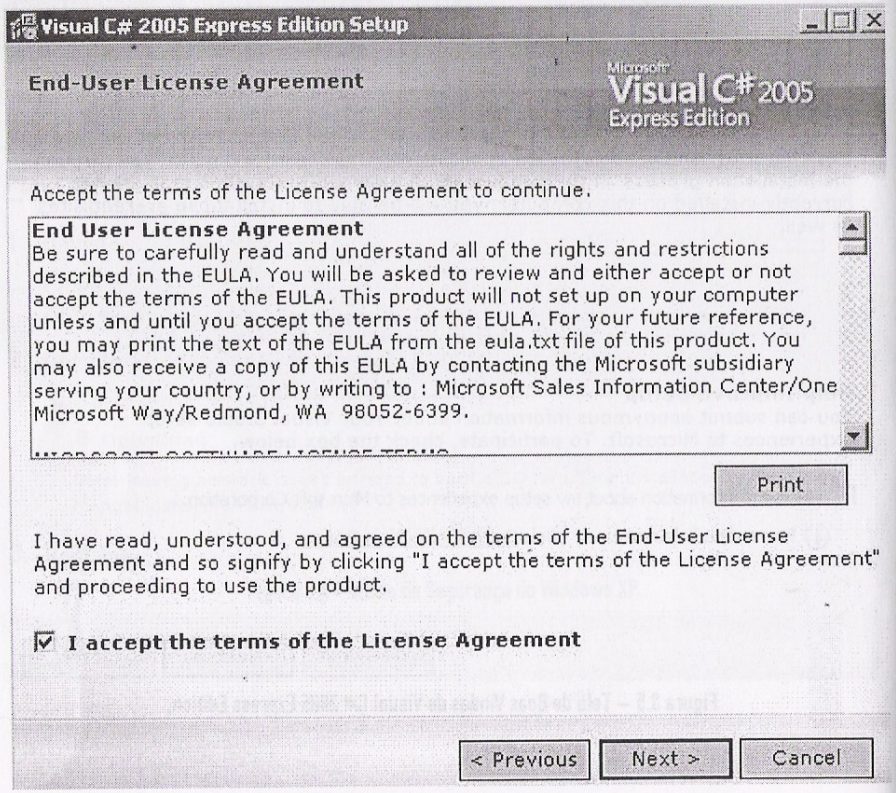


Figura 3.6 –Tela de Licença de uso.

Você pode optar por instalar o MSDN Express Edition Library, que traz uma grande biblioteca de informações, e pode decidir por instalar também o SQL Server 2005 Express Edition. Para isso, basta clicar nas CheckBox de ambos os produtos (figura 3.7).

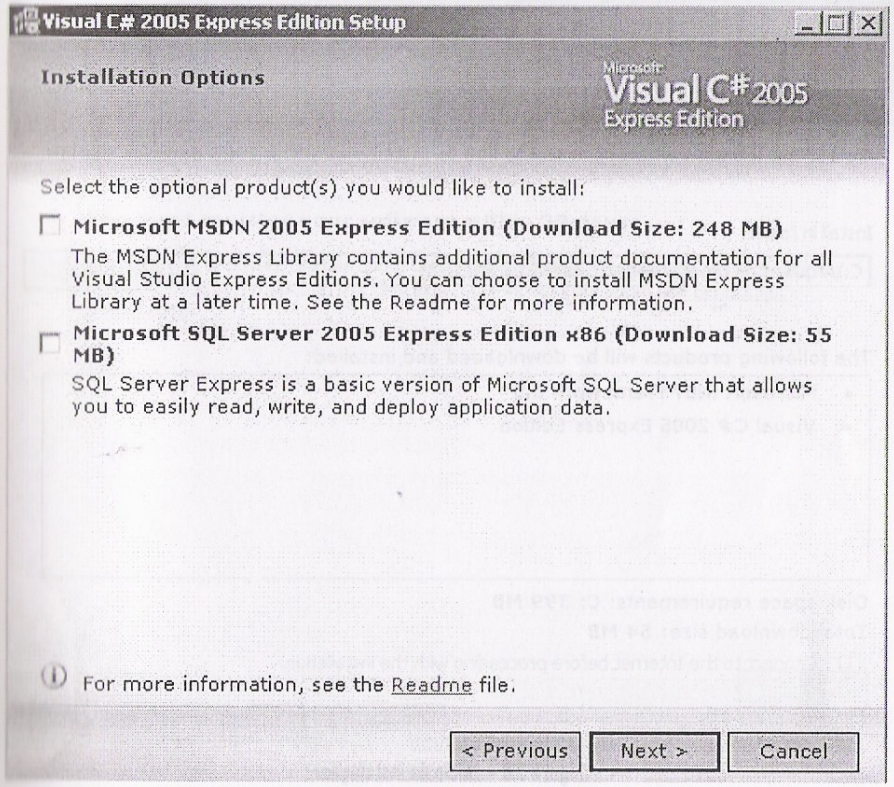


Figura 3.7 –Tela de opções de produtos.

Defina o local da instalação do seu Visual C# 2005 Express Edition. Note que na lista de produtos que serão instalados o Microsoft .NET Framework 2.0 já aparece automaticamente. (Figura 3.8).

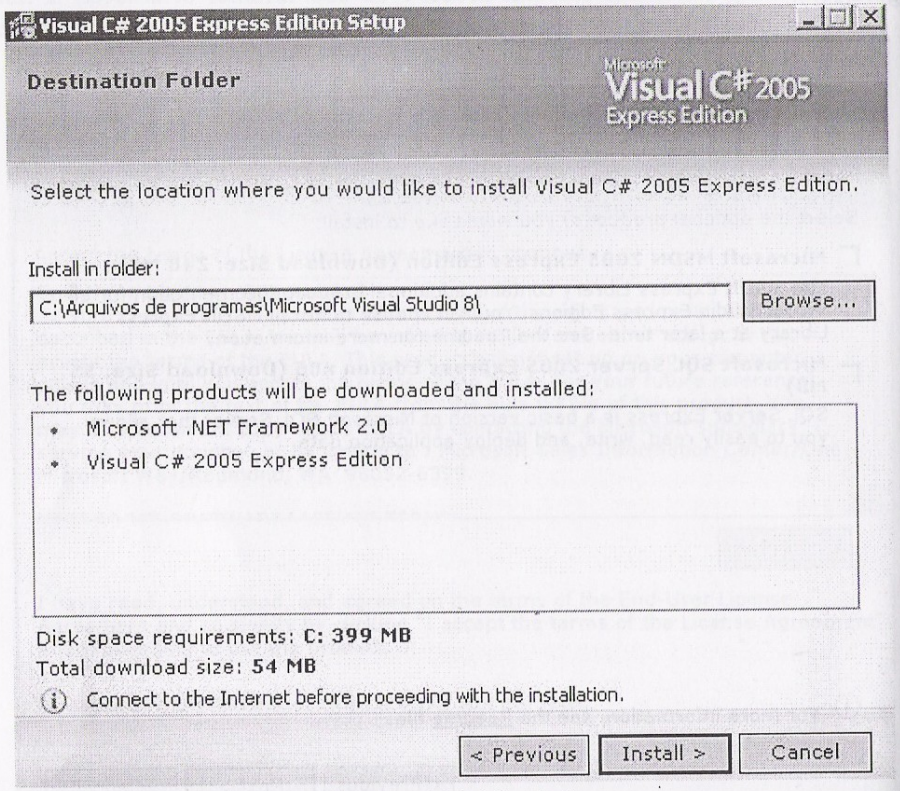


Figura 3.8 – Inicio da instalação.

Ao final da instalação, que não deverá levar mais que 10 minutos, dependendo é claro, da velocidade de seu computador, você receberá a tela abaixo (figura 3.9).

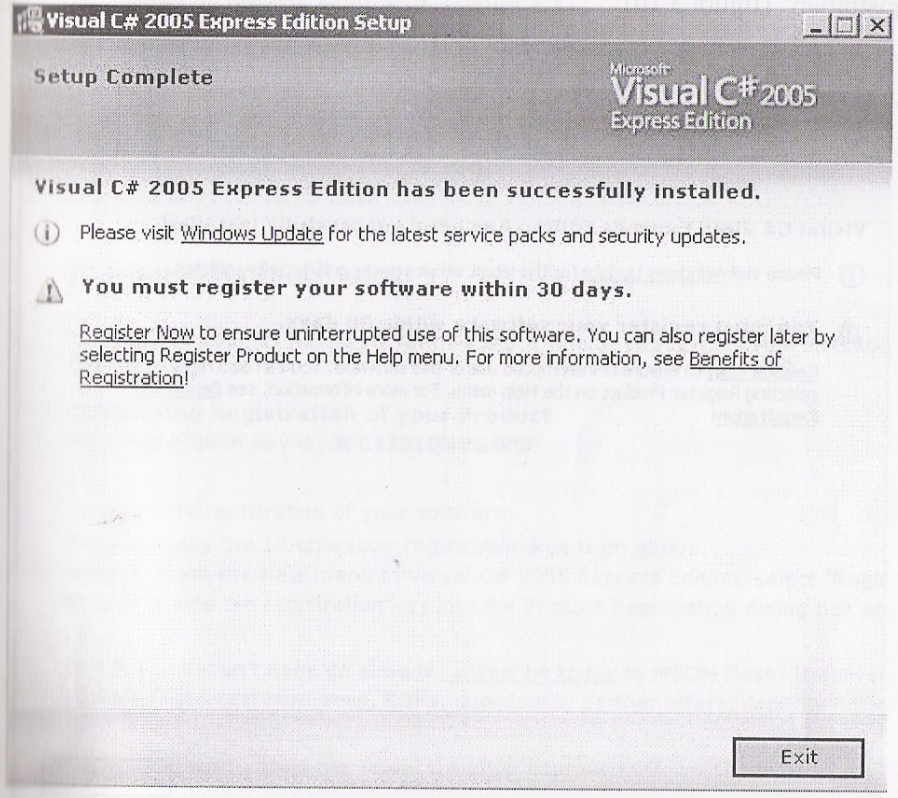


Figura 3.9 – Finalização da instalação.

Antes de clicar em "Exit", clique sobre o link "Register Now" e registre sua cópia. Para saber quais os benefícios em registrar uma cópia, clique no link "Benefits of Registration" (figura 3.10).

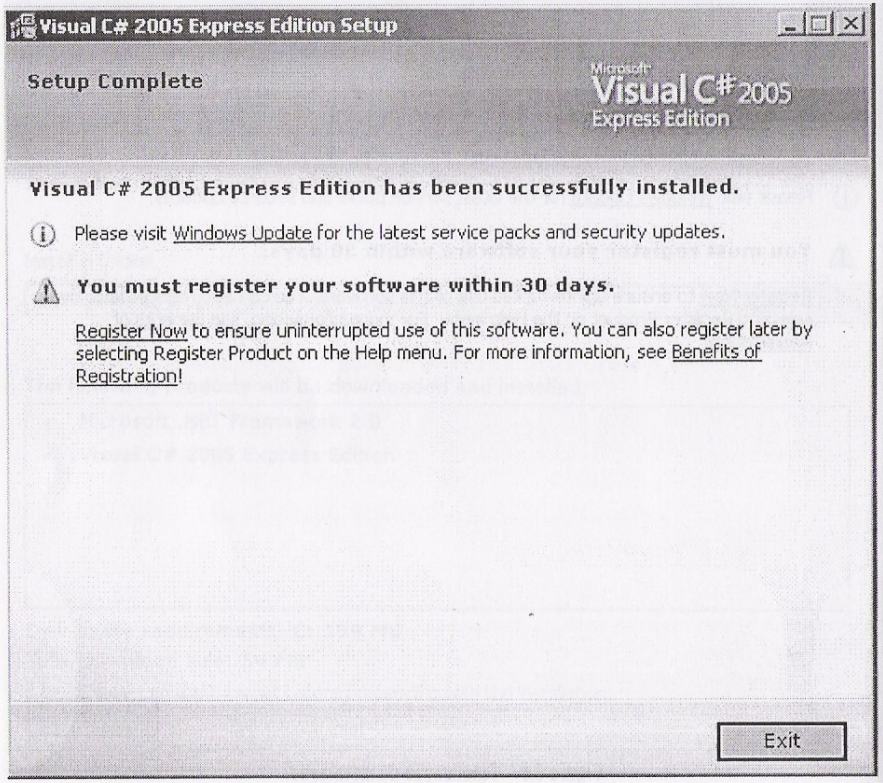


Figura 3.10 – Término da instalação.

Registrando sua cópia, além de contribuir para um melhor controle por parte da Microsoft, você ainda recebe benefícios diversificados que podem variar de acordo com as campanhas da própria Microsoft. (figura 3.11)

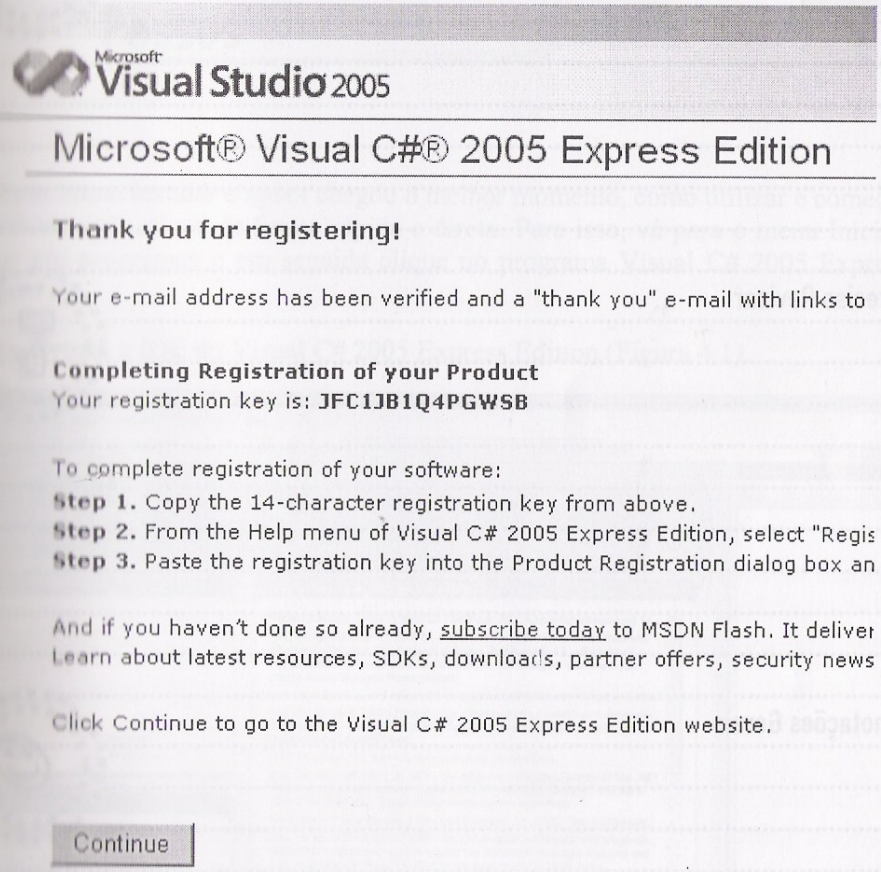
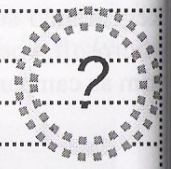


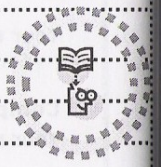
Figura 3.11 – Registro

Nota: Não se esqueça que, após efetuar o registro, quando abrir seu Visual C# 2005 Express Edition pela primeira vez será solicitada a "registration Key" descrita na figura 3.11.

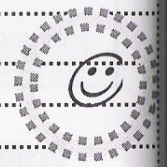
Anotações de Dúvidas



Preciso Revisar



Anotações Gerais



CAPÍTULO 4

C# Express Edition – Primeiros Passos

Já vimos como instalar e agora chegou o melhor momento, como utilizar e começar a produzir aplicativos de forma rápida e direta. Para isso, vá para o menu Iniciar, clique em programas e em seguida clique no programa Visual C# 2005 Express Edition.

Abaixo temos a IDE do Visual C# 2005 Express Edition (Figura 4.1).

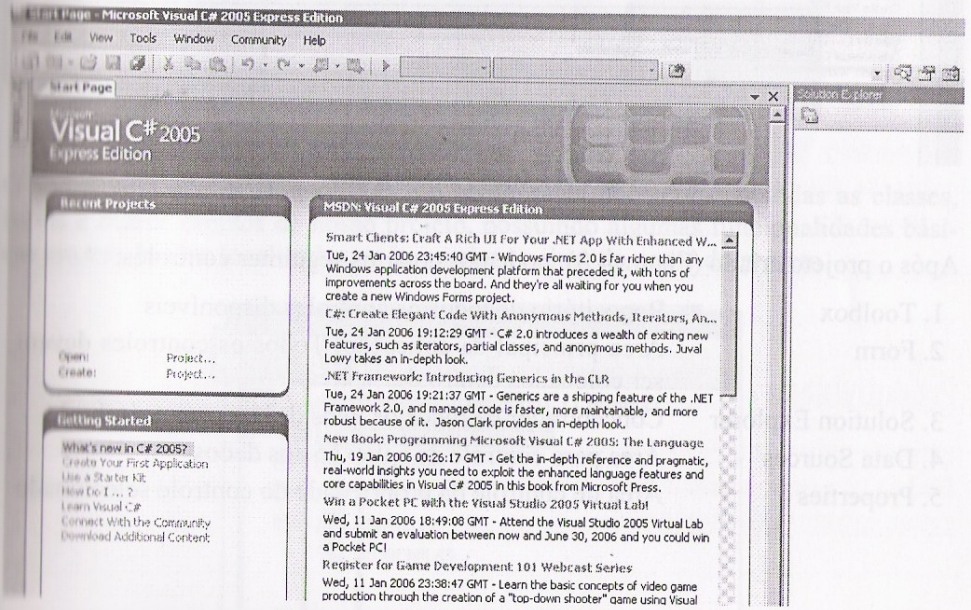


Figura 4.1 – IDE do Visual C# 2005 Express Edition

Principais recursos da IDE

Clique no File, New Project e depois em Windows Project (Figura 4.2) para criar um projeto novo do tipo Windows Forms.

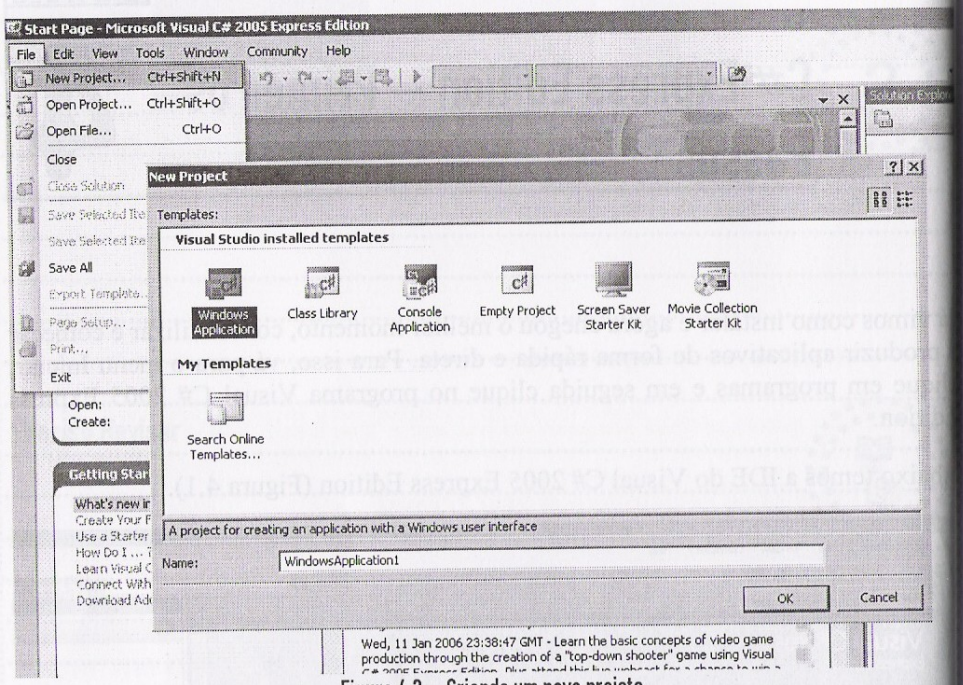


Figura 4.2 – Criando um novo projeto

Após o projeto criado (Figura 4.3), podemos notar os seguintes controles:

1. Toolbox	Repositório de todos os controles disponíveis
2. Form	Área principal da aplicação. Todos os controles devem ser colocados dentro dessa área
3. Solution Explorer	Concentrador de todos os itens de seu projeto/solução
4. Data Sources	Área para controle de conexão aos dados
5. Properties	Área de controle da propriedade do controle selecionado

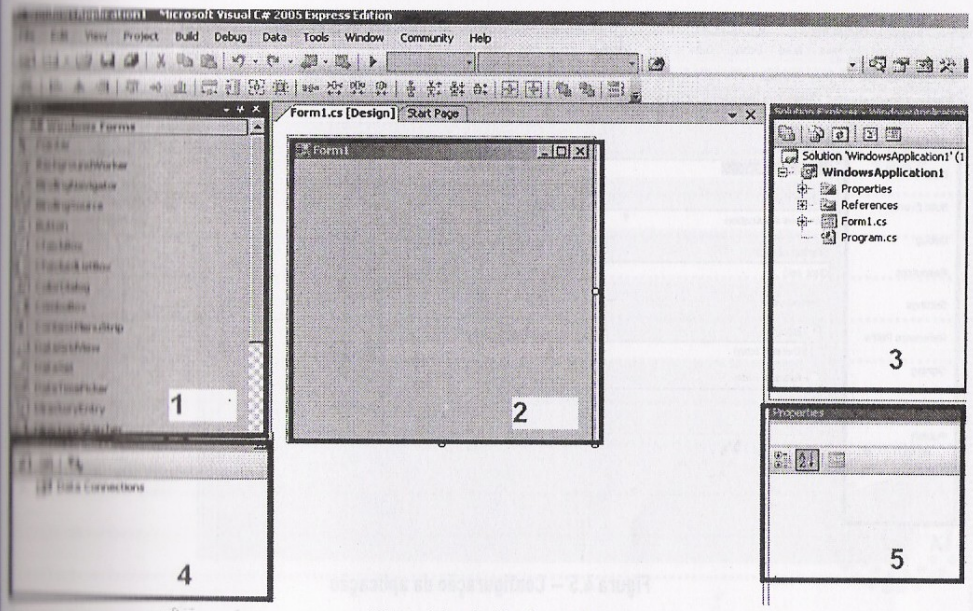


Figura 4.3 – Início do projeto

O Solution Explorer (Figura 4.4) é o responsável por gerenciar todas as classes, forms e outros objetos de nosso projeto, possuindo algumas funcionalidades básicas em três botões localizados na barra cinza acima do nome do projeto.

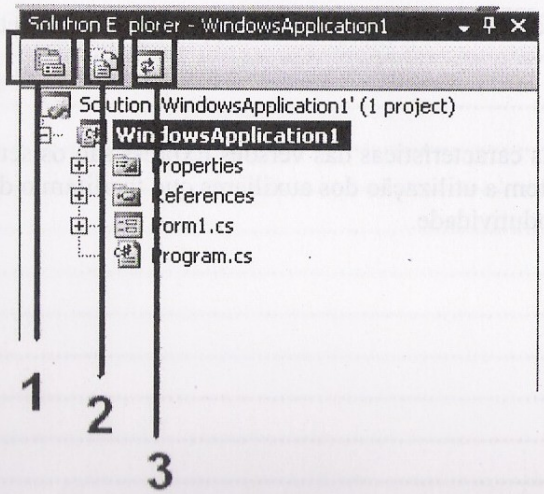


Figura 4.4 – Solution Explorer

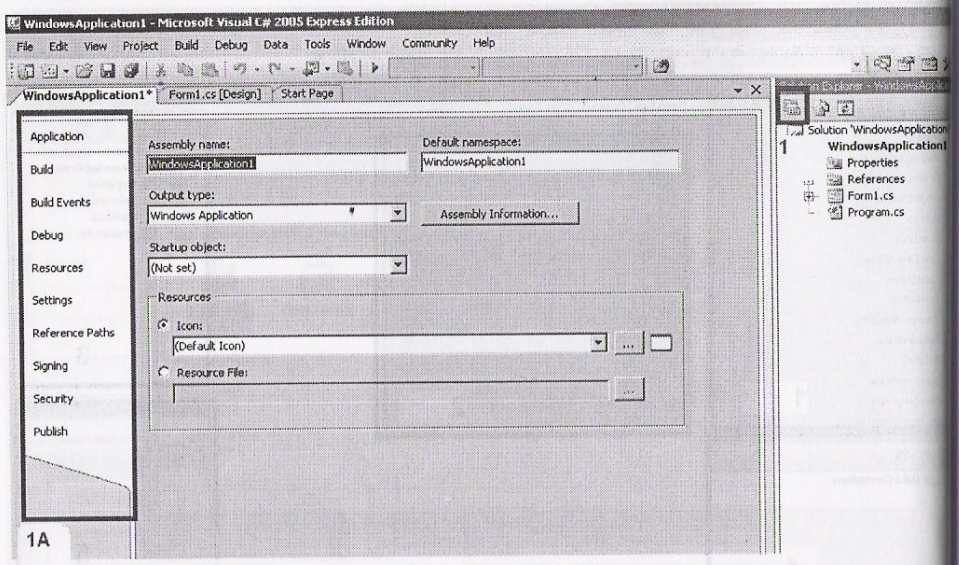


Figura 4.5 – Configuração da aplicação

No Botão 1 (Figuras 4.4 e 4.5) podemos controlar toda a configuração de nossa aplicação (Figura 4.5), desde o nome da aplicação, tipo de autenticação, diretório de debug etc. (1A)

No Botão 2 (Figura 4.4) controlamos a exibição ou não de arquivos e diretórios ocultos.

No Botão 3 (Figura 4.4) temos a atualização dos objetos do projeto.

Produtividade

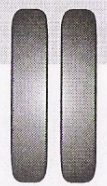
Uma das principais características das versões Express são os seus controles Data Bound, que permitem a utilização dos auxiliares que facilitam o desenvolvimento e geram extrema produtividade.

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

PARTE



Essencial C#

CAPÍTULO 1

Variáveis

O processo de trabalho e manipulação de informações em memória é facilitado com o elemento abstraído conhecido como variável.

A variável é definida por seu tipo e nome (ou rótulo) e está associada a uma área da memória.

Para declarar uma variável em C#, você deverá definir o tipo de dado que será armazenado.

As regras para os tipos de dados compreendendo desde a declaração, passando pelo uso, até o gerenciamento, são definidas pelo **CTS** (Common Type System). Todas as linguagens .NET compartilham o mesmo sistema de tipos (**CTS**) de maneira que um tipo inteiro no C# é exatamente igual ao inteiro do VB.NET.

Para nomear uma variável, você deverá seguir algumas regras:

- ☑ poderá conter letras, números e *underscore* _
- ☑ máximo de 255 caracteres

Veja a sintaxe de declaração em VB.NET:

```
Tipo NomedaVariavel [atribuição]

Exemplos:
string Nome;
string Linguagem = "C#";
string Nome, Endereco, Email;
```

Tipos simples de valor em C#

Inteiros

Tipo	Tamanho bytes	Descrição
Byte	1	(0 a 255) não permite valores negativos
Short	2	(-32,768 a 32,767)
Integer	4	Inteiro com sinal, valores entre -2,147,483,648 e 2,147,483,647
Long	8	Inteiros com sinal, desde -9,233,372,036,854,775,808 até 9,233,372,036,854,775,807

Exemplos de declaração de inteiros:

```
byte Idade = 31;
short Tamanho;
int Pontos = 6000;
long PopulacaoFormigas;
```

Decimais

Tipo	Tamanho Bytes	Descrição
Float	4	Valores de ponto flutuante -3.402823x 10^38 até 1.401298 x 10^-45 (negativos) e 1.401298x10^-45 até 3.402823x10^38 (positivos)
Double	8	Valores de ponto flutuante -1.79769313486231570E+308 até 4.94065645841246544E-324 (negativos) 4.94065645841246544E-324 até 1.79769313486231570E+308 (positivos)

Exemplos de declaração de decimais:

```
float PontodeImpacto = -1.232445646;
double FatorConversao = 4.9993442;
```

Strings e Caracteres

Tipo	Tamanho Bytes	Descrição
Char	2	Caractere Unicode
String	Limitado à plataforma	Armazena cerca de 2 bilhões de caracteres

Exemplos de declaração de Strings e Caracteres:

```
char Opcao = "X";
string Pais = "Brasil";
```

Booleano

Tipo	Tamanho Bytes	Descrição
Boolean	2	Tipo booleano (True, False)

Exemplos de declaração de Booleanos:

```
boolean Confirma;
boolean Confirma = True;
```

Data

Tipo	Tamanho Bytes	Descrição
DateTime	8	Tipo Data

Exemplos de declaração do tipo DateTime:

```
DateTime DtNasc;
DateTime DtInicio = System.DateTime.Now;
```

Vetores

Os vetores, conhecidos tecnicamente como *arrays*, são elementos dimensionais de um determinado tipo de dados, acessados através de um índice.

Exemplos:

```
// Aloca 30 posições do tipo String
// de 0 a 29
string[] Categoria = new string[30];
Categoria[0] = "Livros";
Categoria[1] = "CDs";
Categoria[2] = "DVDs";
...
Categoria[29] = "Eletrônicos";


// Vetor Multidimensional
string[,] Veículos = new string[3,5];
Veiculo[0,0]="Fiat Stilo";
Veiculo[0,1]="Fiat Palio";
Veiculo[1,0]="Vectra";
```

```
Veiculo[1,1]="Omega";
Veiculo[1,2]="Astra";
Veiculo[2,0]="Fox";
Veiculo[2,1]="Gol";
Veiculo[3,0]="Eco Sport";
Veiculo[3,1]="Mondeo";
Veiculo[3,2]="Ford Ranger";

// Vetor de Vetor
string[][] Arvores = new string[][];
```

Constantes

Constantes são valores fixos declarados, que não sofrem qualquer tipo de mudança na execução da aplicação.

Exemplos:

```
const int QTDEMAXIMA = 20;
const int LINHASPORPAGINA = 66;
```

Operadores

Operador	Utilização	Exemplo
=	Atribui um valor a uma variável ou constantes	Pontos = 20;
+	Soma dois valores	Pontos = Pontos + 3;
-	Subtrai um valor de outro	Valor = Pontos – 5;
*	Multiplica dois valores	Resultado = x * y;
/	Divide um valor por outro	Resultado = x / 2;
%	Divide um valor por outro e retorna o resto da operação	Resultado = x % 3;
+	Concatena duas strings	nome = "José" + Sobrenome ;
+=	Soma o valor e atribui o resultado	x += 5;
-=	Subtrai o valor e atribui o resultado	x -= 6;
*=	Multiplica o valor e atribui o resultado	X *= 2;
/=	Divide o valor e atribui o resultado	X /= 2 ;
%=	Divide o valor, obtendo o resto da operação	x %= y;
&=	Concatena a string e atribui o resultado	x &= ", José"
Sqrt	Extrai a raiz quadrada de um valor	x = Math.Sqrt(25)

Enumerações

Enumerações são conjuntos de dados do tipo primitivo, organizados por meio de uma nomeação. As enumerações são fortemente utilizadas para organizar e fornecer informações dentro de uma aplicação.

Vejamos a sintaxe de sua declaração:

```
public (private ) Enum NomedaEnumeracao
   Informacao
   Informacao
   Informacao
   ...
   Informacao
End Enum
```

Exemplos práticos de enumerações

```
public enum TipoConsulta
  {
     Cliente,
     Produto
  }


public enum Cores
        {
            Verde = 0,
            Amarelo = 1,
            Azul = 2,
            Branco = 3
        }
```

Sempre que utilizamos diversos tipos de dados (*datatypes*) temos que nos preocupar com as conversões de valores. Seguem algumas dicas e tabelas em relação a este assunto.

Alguns tipos não podem sofrer conversão. Não podemos, por exemplo, converter uma **string** "*Ferramentas Express*" para **integer**. Os tipos a serem convertidos devem seguir o mesmo padrão lógico, ou seja, números com números, caracteres com caracteres e data com data. Somente no caso de caractere podemos converter praticamente todos os tipos de dados. Podemos converter um número 123 para string, por exemplo.

No caso de valores, nunca podemos converter um tipo de dado **Long** para **Integer**, isto porque o tipo **Integer** possui 4 Bytes e o **Long** 8 Bytes; assim, quando tentamos converter estes tipos de dados, se o campo for maior do que 4 bytes, teremos um grande problema. Inverso a este caso, numa conversão de um tipo **Integer** para um **Long** não existe nenhum problema, ou seja, nunca converta um tipo numérico onde a quantidade de Bytes do valor a ser convertido seja maior do que a queremos converter.

Segue a tabela com todas as possibilidades de conversões:

Tipo	*Possibilidades de conversão*
Byte	Byte, Short, Integer, Long, Decimal, Float, Double
Short	Short, Integer, Long, Decimal, Float, Double
Integer	Integer, Long, Decimal, Float, Double
Long	Long, Decimal, Float, Double
Decimal	Decimal, Float, Double
Float	Float, Double
Double	Double
Char	Char, String
Qualquer tipo	Object

Exemplos de conversão

```
// Convertendo um valor Integer para String
int intValor = 1234;
string strValor;

// Converte o valor para string
strValor = intValor.ToString();


// ------------------------------------------

// Convertendo um valor Decimal para Int
decimal decValor;
int     intValor;

decValor = 35.29;

// Converte o valor para decimal
intValor = (int)decValor;
```


Funções para tratamento de Strings

Existem momentos em que precisamos tratar *Strings*, seja para extrair parte de seu conteúdo, obter o tamanho ou até mesmo converter a cadeia de caracteres entre maiúsculos e minúsculos. Para tanto o C# disponibiliza algumas funções.

Método/Função	*Descrição*
Length	Retorna o comprimento (tamanho) de uma string/array/etc.
Substring	Retorna parte de uma string a partir de uma determinada posição.
toLowe();	Converte os caracteres de uma string para caracteres minúsculos.
toUpper();	Converte os caracteres de uma string para caracteres maiúsculos.
Trim()	Remove todos os espaços ou caracteres específicos contidos em uma string.

Exemplos de uso das funções

```
iTam = Len("Show de Bola");
// iTam recebe valor  12

// ------------------------------------------
string sParte = "Show de Bola";

sParte = sParte.Substring(1,4);
// retorna valor  "Show"

sParte = sParte.Substring(9,4);
// retorna valor  "Bola"
```

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

CAPÍTULO 2

Estrutura Condicional

Tomada de Decisão

A estrutura condicional por tomada de decisão avalia uma expressão lógica e executa um conjunto de instruções caso a expressão seja verdadeira. Opcionalmente, outro conjunto de instruções poderá ser executado caso a expressão lógica não seja verdadeira.

A Figura 6.1 ilustra um caso simples de estrutura condicional.

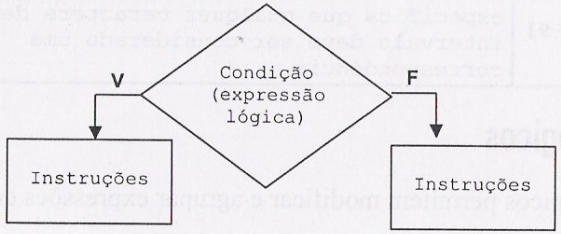


Figura 6.1 – Estrutura Condicional

Em C# temos a seguinte sintaxe para este tipo de estrutura condicional:

```
if (Expressao Lógica 1)
{
   Conjunto de Instruções
}
else if (Expressão Lógica 2) {
    Conjunto de Instruções 2
}
else
{
    Conjunto de Instruções 3
}
```

Operadores de comparação

operador	Descritivo
>	maior que
<	menor que
==	igual a
!=	diferente de
>=	maior ou igual a
<=	menor ou igual a
Like	verifica correspondência de padrões entre strings

O operador Like utiliza caracteres coringas para realizar comparativos lógicos:

Caractere	Função
*	indica a quantidade de caracteres adicionais
?	representa um único caractere
#	representa um dígito {0-9}
intervalos [0-9] , [a-z] , etc.	especifica que qualquer caractere dentro do intervalo deve ser considerado uma correspondência

Operadores Lógicos

Os operadores lógicos permitem modificar e agrupar expressões condicionais.

Operador	*Descrição*
&	O operador booleano AND funciona como a palavra "e", fornecendo a interseção.
||	O operador OR funciona como a palavra "ou", mostrando a união dos conjuntos.
XOR	Gera um resultado verdadeiro se um dos valores/expressão for verdadeiro e o outro for Falso.
NOT	Operador de negação e retorna o oposto a qualquer valor que for usado com ele.

Exemplos de tomadas de decisão

```
int x = 10;
int y = 11;
int fator;

// verifica se x é igual a y
if (x == y) {
 Fator = 1;
} else {
 Fator = -1;
}

// ---------------------------------

string Linguagem = "C#";
string Resposta;

// verifica se Linguagem é diferente de "C#"
 if (Linguagem != "C#") {
 Resposta = "Wrong Choice!";
}
// ---------------------------------
int Valor1;
int Valor2;
int Valor3;

// verifica se Valor1 é igual a Valor 2 e Valor1 menor que
Valor3

if ((Valor1 == Valor2) & (Valor1 < Valor3)) {
 Valor1 = Valor3;
}
// ---------------------------------
int x;
int y;
int z;

// verifica x é diferente de y OU se x = z
if ((x != y) | (x == z)) {
 x = 0;
 y = 0;
 z = -1;
}
```

Tomada de Decisão do Tipo Escolha

A estrutura condicional por tomada de decisão do tipo escolha avalia o valor de uma variável e executa as instruções do bloco correspondente a este valor. A Figura 6.2 ilustra um caso simples de estrutura condicional.

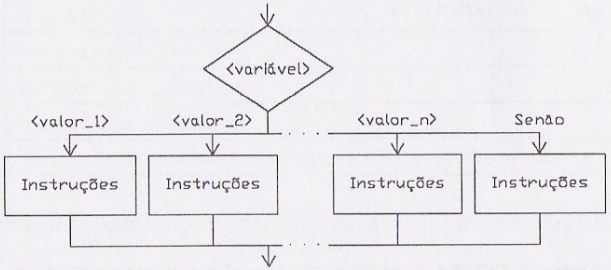


Figura 6.2 – Tomada de decisão do tipo escolha

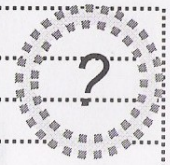
Em C# temos a seguinte sintaxe para este tipo de estrutura condicional:

```
switch (Variável ou Expressão)
  case (Condicao1)
     {
      Rotina1
      break;
     }
  case (Condicao2)
     {
      Rotina2
      break;
     }
  Default
  {
   Rotina se não satisfez as demais

  }
}
```

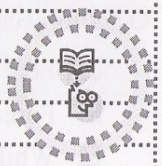
Exemplos de tomada de decisão do tipo escolha

```
switch (opção) {
   case (1)
      {  Console.WriteLine('opcao #1'); break }
   case (2)
      {  Console.WriteLine('opcao #2'); break }
   default
      {  Console.WriteLine('opcao padrao'); break }
}
```


Anotações de Dúvidas



Preciso Revisar



Anotações Gerais

CAPÍTULO 3

Laços

Em muitas situações existe a necessidade de repetir um conjunto de instruções. As linguagens de programação oferecem uma estrutura simples para este tipo de operação, conhecida como laços.

O C# fornece quatro tipos de laços:

- ☑ For
- ☑ Foreach/in
- ☑ While
- ☑ Do/While

Laço For

Repete o bloco de código "n" vezes.

Sintaxe do laço *For*

```
for (variável contador; condição; incremento)
{
   ... bloco de código
}
```

Contador	Variável numérica que controla o número de vezes em que o bloco deve ser executado.
Condição	Expressão para condicionar a execução do laço.
Incremento	Incrementa/Decrementa Contador.

Exemplo de laço *For*

```
// neste exemplo a instrução será executada 5 vezes
```



```
for (int iContador = 0; iContador < 5; iContador++)
{
   Console.WriteLn("Olá meu nobre amigo");
}
```

```
// neste exemplo o laço será interrompido quando o valor dos
pontos ultrapassar 300 ou o contador atingir o número 50

int Pontos = 0;

for (int iContador = 0; iContador < 50; iContador++)
{
   Console.WriteLn(Pontos);
   Pontos += 25;
   If (Pontos == 300) { break; }
}
```

Laço foreach/in

Este tipo de laço é utilizado para varrer arrays ou coleções.

Sintaxe do laço *foreach/in*

```
foreach(elemento in coleção)
{
  // bloco de código
}
```

Exemplo do laço *foreach/in*

```
string [] nome = new string [] {"Alexandre Tarifa", "Emerson
Facunte", "Marcus Garcia"}

foreach(string s in nome)
{

   Console.WriteLine(nome);

}
```

Laço While

O Laço While executa um bloco de código enquanto a expressão booleana for verdadeira.

Sintaxe do laço *While*

```
While (Expressão Booleana)
{
   // [código a ser executado]
}
```

Exemplo do laço *While*

```
int x = 0;

while ( x < 10 )
{
    x++;
}
```

Laço Do/While

O Laço Do/While é extremamente flexível, permitindo a avaliação da expressão booleana no início do laço ou no final do laço. Neste último caso, o laço é executado pelo menos uma vez.

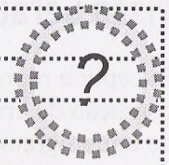
Sintaxe do laço *Do/While*

```
do
{
   //[código a ser executado]


}

while (expressão condicional)
```


Exemplos de laço *Do/While*

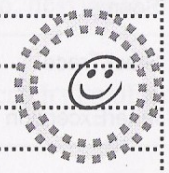
```
//  executa enquanto x < 10
int x
do
{
   x++;
}
while (x < 10);
```

Anotações de Dúvidas



Preciso Revisar

Anotações Gerais



CAPÍTULO 4

Tratamento de Erros – Exception

Visão Geral

Exception representa uma condição excepcional, um comportamento inesperado ou condição de erro, interrompendo o fluxo normal das instruções durante a execução de um programa.

O .NET oferece um mecanismo robusto para tratamento de exceções. Na realidade, tratar exceções é o mesmo que tratar erros com classe, elegância e sofisticação.

Esse poder provém da biblioteca System.Exception.

Membros do objeto Exception:

Propriedade	Descritivo
HelpLink	Retorna uma URL apontando para um documento HELP com o descritivo do erro em detalhes
Message	Descritivo do erro
Source	Nome do objeto ou aplicação que gerou o erro
StackTrace	Apresenta a seqüência de chamadas que dispararam o erro
InnerException	Apresenta exceções aninhadas

Estrutura

Conheça a estrutura básica para tratamento de exceções em C#:

```
using System;
namespace ProjetoExpress
```



```
{
  class MainClass
  {
 public static void Main(string[] args)
  {
      try
      {
      // Código a ser executado
      }
      catch (TipoExceção1 e)
      {
      // tratamento da exceção1
      }
      catch (TipoExceção2 e)
      {
      // tratamento da exceção2
      }
      catch (Exception e) // erro genérico
      {
      // tratamento genérico
      }
      finally
      {
// Este bloco de código sempre será executado, havendo ou não
exceção!
      }
  }
  }
}
```

Quando uma exceção for gerada, apenas um bloco catch é executado, desviando o fluxo para o bloco finally (se este existir).

Digamos que uma exceção do tipo 1 é gerada, então apenas o bloco catch (TipoExceção2) é executado, além do finally.

Gerando suas próprias exceções

Você também poderá gerar suas próprias exceções utilizando o comando throw.

Sintaxe:

```
  // Gerando uma exceção simples
  Throw New Exception("Erro! Idade inferior a 18 anos")
```

```
// Gerando uma exceção baseada numa classe
Public class eIdadeInvalida: Exception {}

...
try
 {
     ...
 }
catch
 {
     Throw new eIdadeInvalida()
 }
```

Exemplos

Neste ponto serão ilustrados alguns exemplos de tratamento de exceções.

Exemplo 1 – Simples exceção

No primeiro exemplo forçaremos um erro de divisão por zero.

```
using System;

namespace ProjetoExpress
{
      class MainClass
      {
            public static void Main(string[] args)
            {
                  try
                  {
                  // Código
                        int x = 5;
                        int y = 0;
                        int z = x/y;
                        Console.WriteLine(z); // esta linha
não será executada

                  }
                  catch (System.Exception e)
                  {
                  // Apresenta o erro
                        Console.WriteLine(e.Message);
                  }
```



```
                  finally
                  {
                  // Este bloco de código sempre será
executado, havendo ou não exceção!
                        Console.WriteLine("Simples, não é?");
                  }
            }

      }
}
```

A Figura 8.1 ilustra o primeiro exemplo sendo executado.

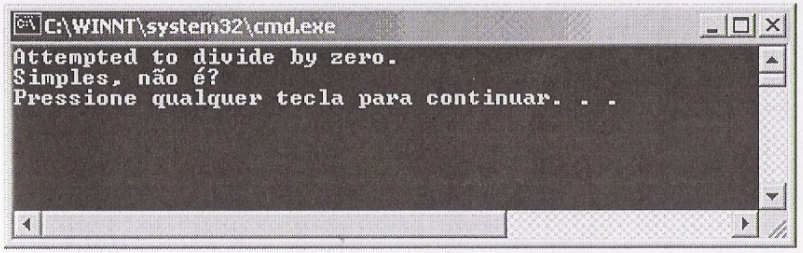


Figura 8.1 – Primeiro exemplo sendo executado

Exemplo 2 – Criando exceção explícita

No segundo exemplo você criará uma exceção explícita.

```
using System;

namespace ProjetoExpress
{
  class MainClass
  {
   public static void Main(string[] args)
   {
     try
      {
      // Código
      int Hora = (System.DateTime.Now.Hour);
      if (Hora>13)
      {
       throw new Exception("Backup não poderá ser realizado
       após as 13 horas!");
```

```
      }

      Console.WriteLine("Ok... Backup realizado");
     // esta linha não será executada se horário > 13:59:59

      }
      catch (System.Exception e)
      {
       // Apresenta o erro
       Console.WriteLine(e.Message);
      }

      finally
      {
       // Este bloco de código sempre será executado, havendo
ou não exceção!
      Console.WriteLine("Simples, não é?");
      }
     }
   }
}
```

A Figura 8.2 ilustra o segundo exemplo em execução.

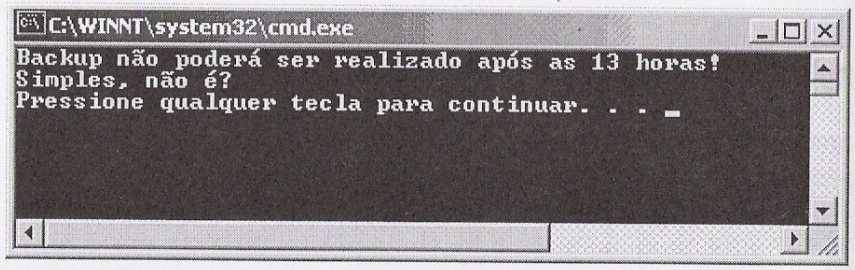


Figura 8.2 – Segundo exemplo sendo executado

Exemplo 3 – Criando novas exceções

No terceiro exemplo você aprenderá a criar suas próprias exceções.

```
using System;

namespace ProjetoExpress
{
 class MainClass
```



```
{
      public class eInvalidTime: Exception {}
      public class eInvalidDay : Exception {}
      public static void Main(string[] args)
      {
       try
       {
        // Código
        int Hora = (System.DateTime.Now.Hour);
        int Dia = (System.DateTime.Now.Day);

        if (Dia ==8) { throw new eInvalidDay();}
        if (Hora<22) { throw new eInvalidTime();}

        Console.WriteLine("Ok... Backup realizado");
       // esta linha não será executada se horário > 13:59:59

       }
       catch (eInvalidTime)
       {
        // Apresenta o erro para o tipo eInvalidTime
        Console.WriteLine("------ erro: InvalidTime");
        Console.WriteLine("O backup não poderá ser realizado
        antes das 22:00h ");
       }
       catch (eInvalidDay){
       // Apresenta o erro para o tipo eInvalidDay
        Console.WriteLine("------ erro: InvalidDate");
        Console.WriteLine("Execute manualmente o
        procedimento de backup");
       }
       finally
       {
        // Este bloco de código sempre será executado,
           havendo ou não exceção!
        Console.WriteLine("Simples, não é?");
       }
      }
 }
}
```

A Figura 8.3 ilustra a aplicação sendo executada no dia 8, às 10:00hs.

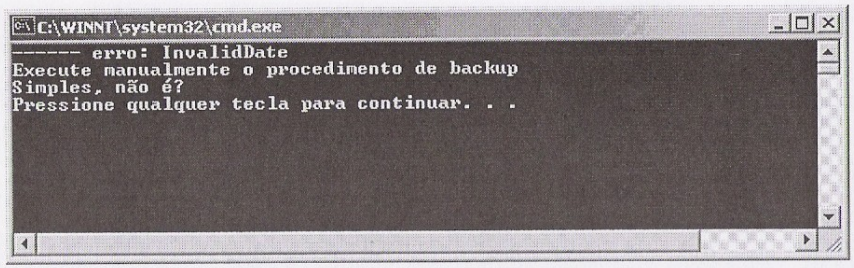
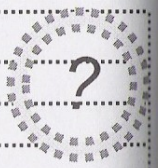
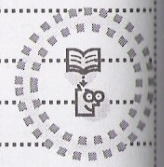


Figura 8.3 – Exceções personalizadas

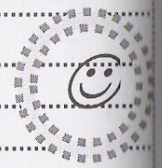
Anotações de Dúvidas



Preciso Revisar



Anotações Gerais



CAPÍTULO 5

Classes

Visão geral

Uma classe define os atributos e os métodos para um conjunto de objetos. Todos os objetos desta classe (as instâncias da mesma) partilham o mesmo comportamento e têm o mesmo conjunto de atributos (cada objeto tem o seu próprio conjunto).

Estrutura da classe em C#

```
public class Cliente
{

  // atributos
  private int _cli_Id;
  private string _cli_NomeRazao;

  // metodos
  public Incluir()
  {
          // codigo
  }

  public Alterar()
  {
          // codigo
  }
  public Excluir()
  {
          // codigo
  }
  public Pesquisar()
  {
          // codigo
  }
}
```

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

PARTE III

Desenvolvimento da Aplicação

CAPÍTULO 1

Negócio

Visão Geral do Projeto

O projeto do livro consiste num sistema de vendas, onde temos os cadastros básicos: cliente, categoria e produto, com todas as funcionalidades (inclusão, consulta, alteração e exclusão de informações), bem como o cadastro de pedidos.

Cadastro de clientes

Cadastro de clientes com informações básicas para realizar pedidos.

Cadastro de categorias

Cadastro de categorias de produtos.

Cadastro de produtos

Cadastro de produtos com informações de valores e quantidade em estoque.

Cadastro de pedidos

Cadastro de pedidos e itens, com base nos cadastros de clientes e produtos.

O objetivo é demonstrar conceitos de orientação a objetos, além do poder das ferramentas Express.

Banco de Dados

Chegamos à etapa de criação do banco de dados.

Iremos utilizar o MS Access para criação do banco de dados de nosso projeto.

Tabela Cliente

Tabela para armazenamento de dados dos clientes.

Nome do Campo	Tipo	Tamanho	Descritivo
cli_Id (PK)	Autonum		Código (Id)
cli_NomeRazao	Texto	80	Nome/Razão social
cli_CNPJCPF	Texto	20	Nº CNPJ ou CPF
cli_Logradouro	Texto	80	Endereço
cli_Bairro	Texto	50	Bairro
cli_Cidade	Texto	50	Cidade
cli_UF	Texto	2	Unidade Federação (Estado)
cli_CEP	Texto	8	CEP
cli_Email	Texto	80	e-mail
cli_Fones	Texto	50	Telefones
cli_DtCadastro	Data/Hora		Data do cadastro

A partir do MS Access, crie um novo banco de dados em branco.

Crie as tabelas[1] a seguir, nomeando-as de acordo com a referência (Cliente, Categoria, Produto...).

Tabela Categoria

Tabela para armazenamento de categorias de produtos.

Nome do Campo	Tipo	Tamanho	Descritivo
cat_Id (PK)	Autonum		Código (Id)
cat_Descricao	Texto	50	Descrição

Tabela Produto

Tabela para armazenamento de dados dos produtos.

Nome do Campo	Tipo	Tamanho	Descritivo
pro_Id (PK)	Autonum		Código (Id)
cat_Id (FK)	Número	int longo	Código Categoria (Id)
pro_Descricao	Texto	50	Descrição
pro_QtdeEstoque	Número	int longo	Quantidade em estoque
pro_Valor	Unid Monetária		Valor
pro_Ativo	Sim/Não		Produto ativo (Sim ou Não)

[1] Para criar uma nova tabela, clique na opção "Criar Tabela em Modo de Design" e preencha os campos conforme segue.

Tabela Pedido

Tabela para armazenamento de dados dos pedidos.

Nome do Campo	Tipo	Tamanho	Descritivo
ped_Id (PK)	Autonum		Número do Pedido (Id)
Cli_Id (FK)	Número	int longo	Código Categoria (Id)
Ped_dtEncomenda	Data/Hora		Data do Pedido
Ped_Valor	Unid Monetária		Valor do Pedido
Ped_Status	Texto	1	Status do pedido

Tabela Item

Tabela para armazenamento de dados dos itens de pedido.

Nome do Campo	Tipo	Tamanho	Descritivo
Ite_Id (PK)	Autonum		Código (Id)
Ped_Id (FK)	Número	int longo	Código do Pedido
Pro_Id (FK)	Número	int longo	Código do Produto
Ite_Qtde	Número	int longo	Quantidade do item
Ite_Valor	Unid. Monetária		Valor do item

Relacionamentos

Para realizar os relacionamentos, selecione a opção "Ferramentas/Relacionamentos" do menu principal.

Para cada relacionamento, clique na tabela consulta e arraste o mouse até a tabela consulta relacionada, informando os dados com segue:

#1

Tabela/Consulta	Tabela/consulta relacionada
Cliente	Pedido
cli_id	cli_id

#2

Tabela/Consulta	Tabela/consulta relacionada
Pedido	Item
Ped_Id	Ped_Id

#3

Tabela/Consulta	Tabela/consulta relacionada
Produto	Item
pro_id	pro_id

#4

Tabela/Consulta	Tabela/consulta relacionada
Categoria	Produto
Cat_id	Cat_id

A Figura 10.1 ilustra o relacionamento completo das tabelas de nossa aplicação.

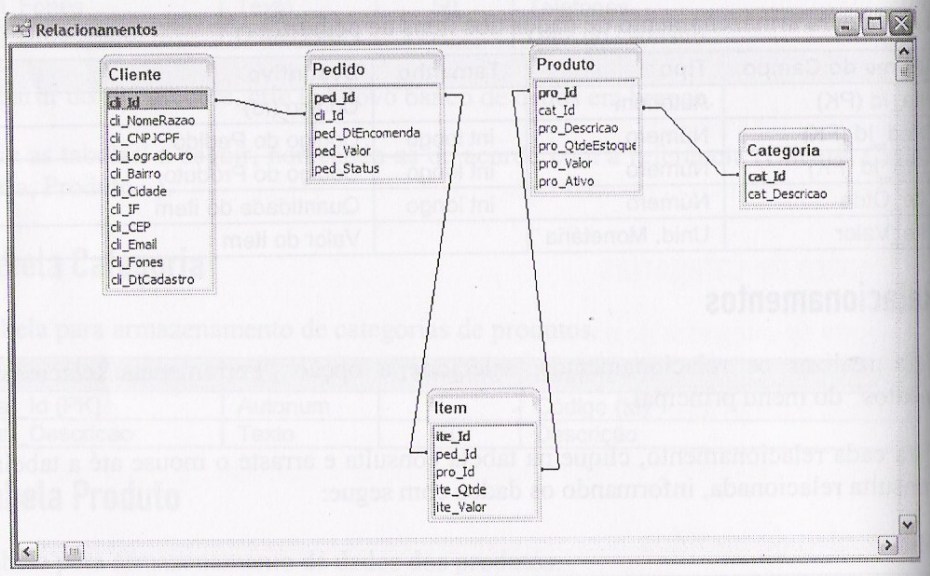


Figura 10.1 – Relacionamentos

Diagrama de classes

O sistema é organizado em classes (Figura 10.2) facilitando o entendimento e o desenvolvimento da aplicação.

O diagrama de classes ilustra – sob o ponto de vista lógico – as especificações para classes, com suas associações e atributos, além dos métodos, operadores e constantes.

Para o desenvolvedor, o diagrama representa os papéis e as responsabilidades das entidades que fornecem o comportamento do sistema.

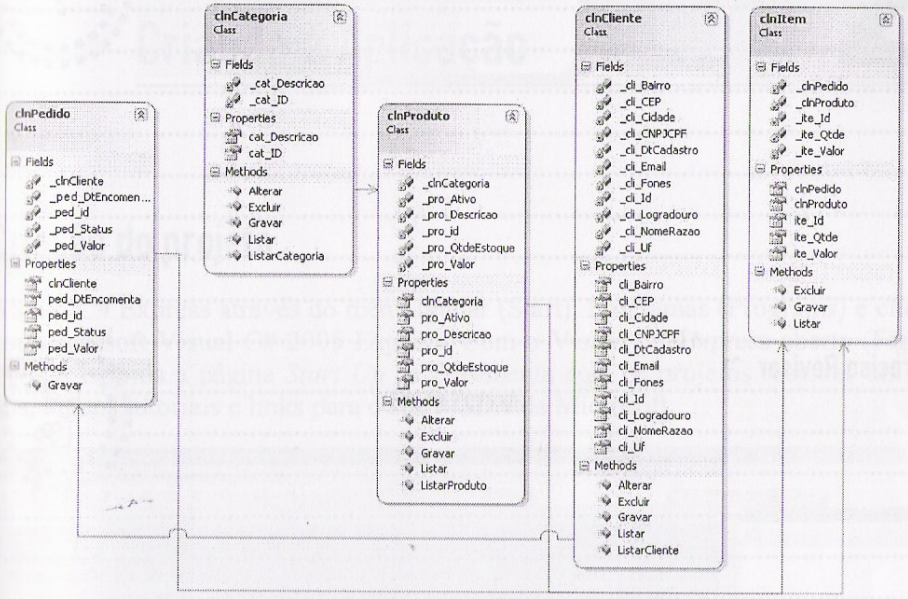
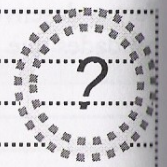
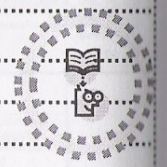


Figura 10.2 – Diagrama de Classes do Sistema

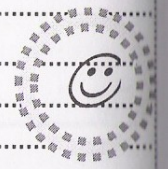
Anotações de Dúvidas



Preciso Revisar



Anotações Gerais



Criando a Aplicação

Criação do projeto

Abra o C# Express através do menu Iniciar (Start), Programas (Programs) e clique em Microsoft Visual C# 2005 Express. Com o Visual C# Express aberto (Figura 11.1) é exibida a página *Start Up* da ferramenta com os projetos recentes utilizados, alguns tutoriais e links para o site *MSDN* da Microsoft.

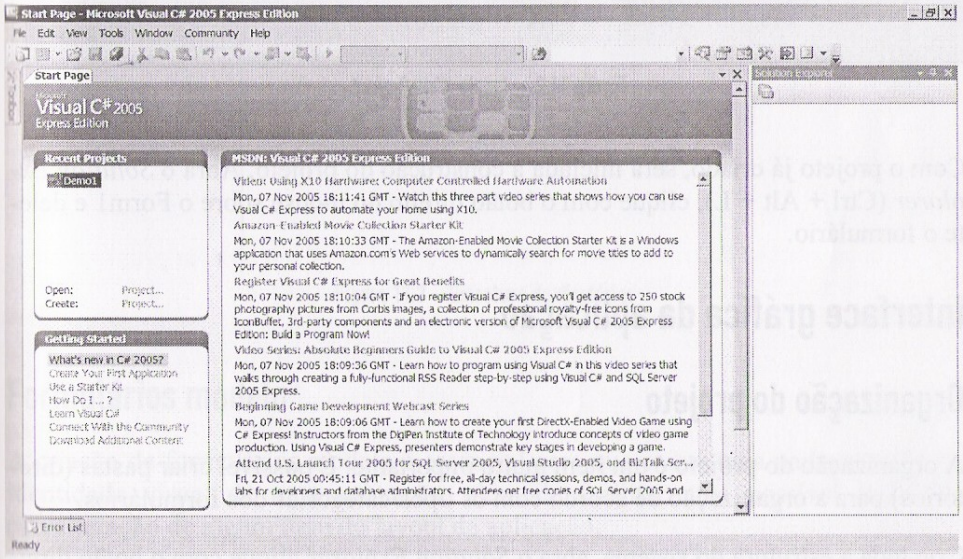


Figura 11.1 – C# Express – tela inicial

No menu, clique em File, New Project e a janela *New Project* (Figura 11.2) é exibida. Selecione o template Windows Application e defina o nome como ControleDeVendas e clique no botão OK.

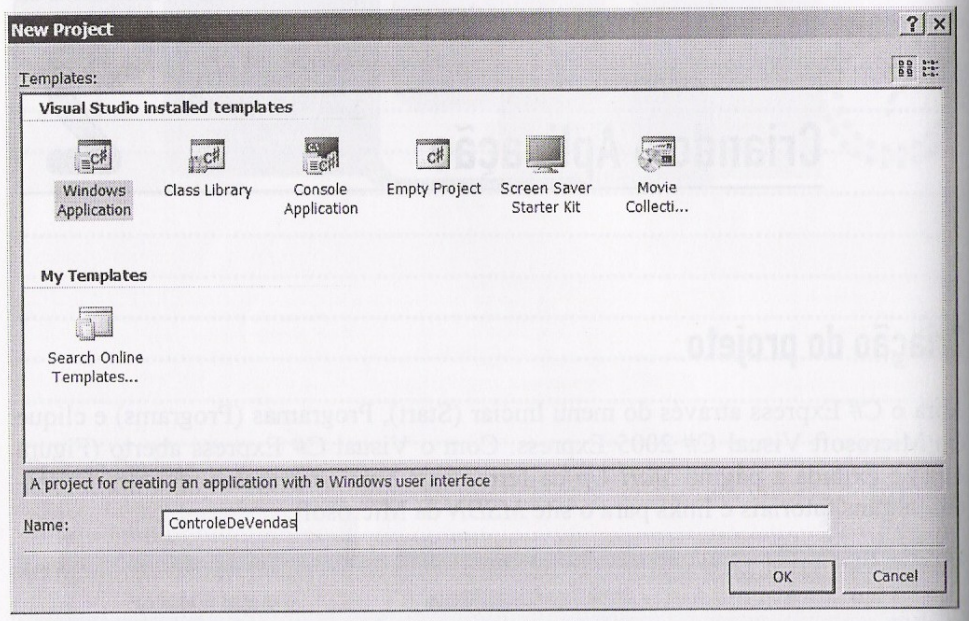


Figura 11.2 – Janela *New Project*.

Com o projeto já criado, será iniciada a construção do projeto. Abra o *Solution Explorer* (Ctrl + Alt + L), clique com o botão direito do mouse sobre o Form1 e delete o formulário.

Interface gráfica da aplicação

Organização do projeto

A organização do projeto é um item muito importante. É possível criar pastas (diretórios) para a organização de acordo com os tipos/categorias dos formulários.

Crie toda a estrutura do sistema, abra o *Solution Explorer* clique com o botão direito do mouse sobre o projeto, Add, New Folder. A criação é semelhante ao Windows Explorer. Para alterar os nomes de diretórios, basta selecionar o diretório com o botão direito do mouse e clicar em Rename.

Crie toda a estrutura conforme a Figura 11.3.

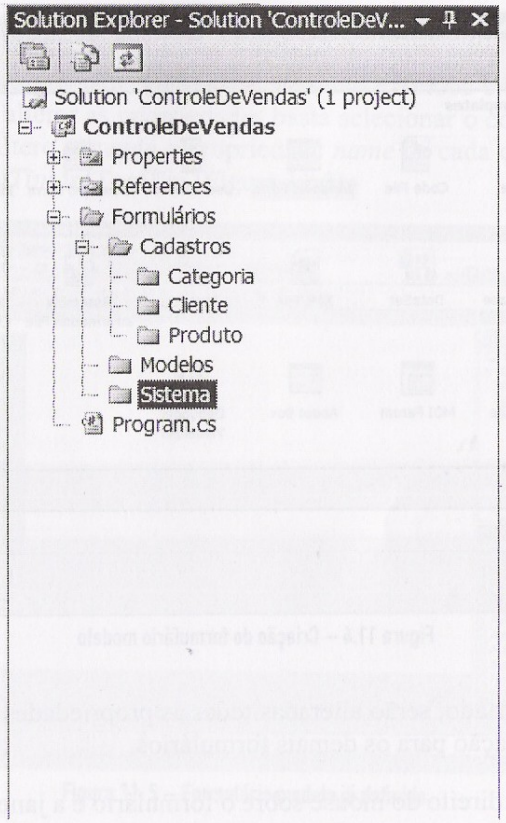


Figura 11.3 – Estrutura de diretórios

Formulários modelo

A criação de formulários modelo é essencial para que exista uma padronização da identidade visual de uma aplicação, além de facilitar uma possível manutenção/alteração de algum item do layout da aplicação.

É possível criar um formulário com propriedades e funcionalidades que serão reaproveitadas por outros formulários, além da possibilidade de definição do encapsulamento (permissão de acesso).

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Modelos, Add e Windows Form. Uma nova janela é aberta (Figura 11.4) para a seleção do template (Windows Form) e o nome, frmModelo e clique em OK.

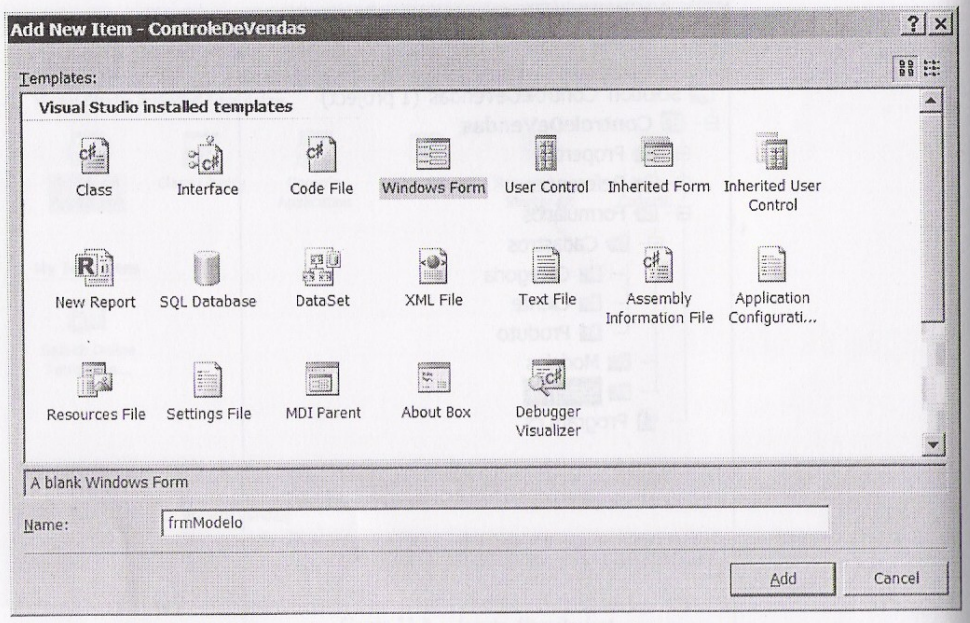
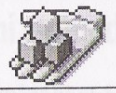


Figura 11.4 – Criação do formulário modelo

Com o formulário criado, serão alteradas todas as propriedades e definida a identidade visual da aplicação para os demais formulários.

Clique com o botão direito do mouse sobre o formulário e a janela de propriedade é aberta para a definição. Altere as propriedades conforme segue.

OBJETO		
	Form	
Objeto	**Propriedade**	**Valor**
frmModelo	Font	Verdana
	ShowIcon	False
	ShowInTaskBar	False
	Text	Em branco

Além de recursos de layout é possível adicionar controles no formulário modelo para que todos os formulários que utilizem o modelo não necessitem adicionar os controles a todo o momento. Serão utilizados dois controles em praticamente todos os formulários do sistema; são eles *ErrorProvider* e *ToolTip*, que serão explicados

no decorrer da aplicação. Adicione os dois controles no formulário utilizando o *ToolBox* (Ctrl + Alt + X), selecione os controles e arraste para dentro do formulário. Como esses controles não são visuais, eles são exibidos em uma barra abaixo do formulário. Para alterar as propriedades basta selecionar o controle com o botão direito do mouse. Altere somente a propriedade *name* de cada controle: *ErrorProvider*: errErro e *ToolTip*: totToolTip (Figura 11.5).

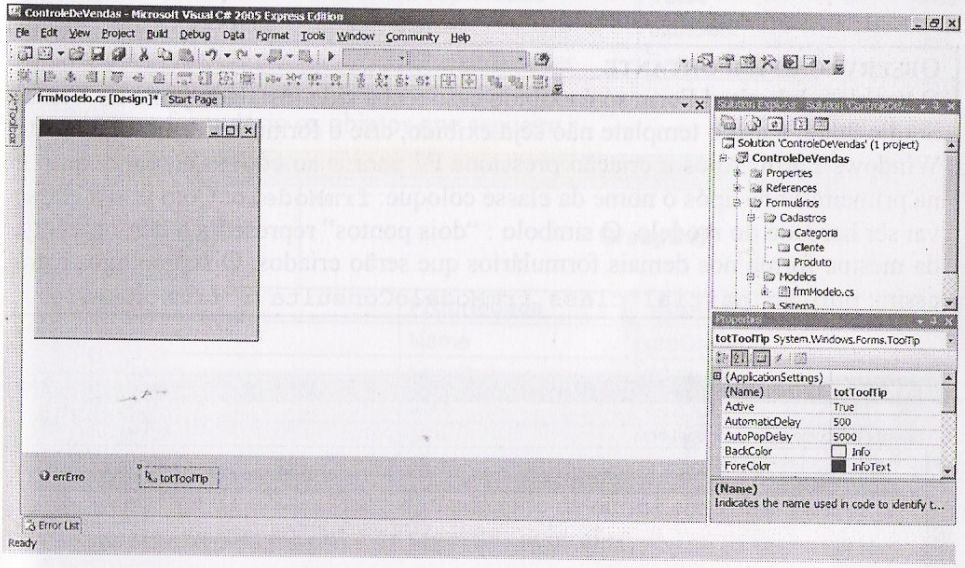


Figura 11. 5 – Formulário modelo já definido

É necessário compilar a aplicação para utilizar o modelo nos próximos formulários. No *Solution Explorer*, dê um duplo clique na classe *Program* localizado no *Solution Explorer* e altere na classe `Application.Run(new Form1());` para `Application.Run(new Formulários.Modelos.frmModelo());`. A classe *Program* é responsável por eventos e configurações do projeto. No menu *Build* clique em *Build Solution*.

Basicamente na aplicação teremos dois tipos de formulário: cadastros e consultas. Todas as telas de cadastros possuem características em comum, assim como as de consulta, portanto criar sobre o formulário modelo outros modelos facilita muito o desenvolvimento. Serão criados dois novos modelos: Modelo Cadastro e Modelo Consulta.

Modelo de Consulta

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Modelos, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmModeloConsulta e clique em OK. Uma nova janela é aberta (Figura 11.6) para a seleção do formulário modelo, selecione e clique em OK.

> **OBSERVAÇÃO IMPORTANTE**
> O template Inherited Form só é exibido caso tenha sido instalado junto ao Visual Studio 2005. Caso o template não seja exibido, crie o formulário com o template Windows Form. Após a criação pressione F7 para ir ao código do formulário e na primeira linha após o nome da classe coloque: `frmModelo`. Com isso a classe vai ser herdada do modelo. O símbolo : "dois pontos" representa a herança. Siga da mesma forma nos demais formulários que serão criados. O seu código ficará assim: `public partial class frmModeloConsulta : frmModelo`

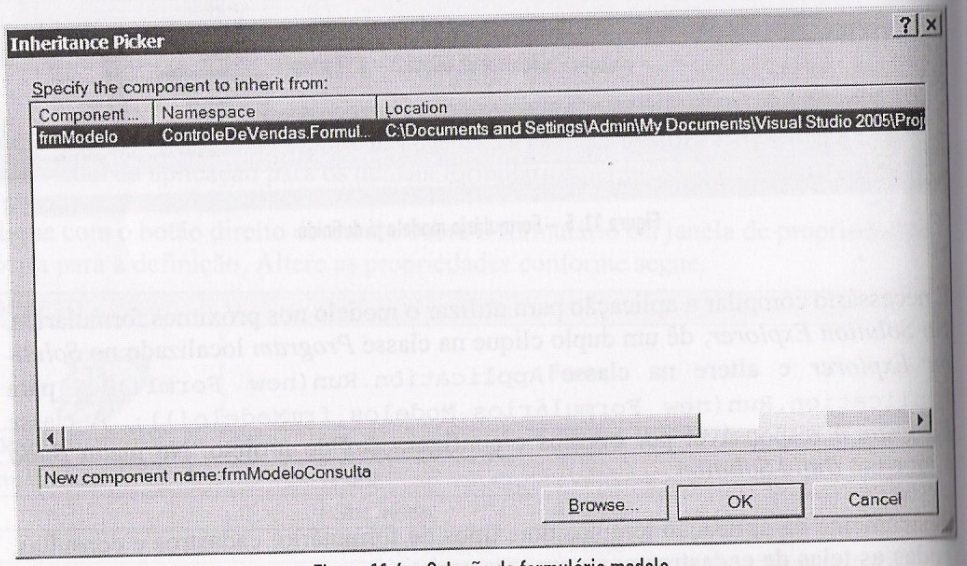


Figura 11.6 – Seleção do formulário modelo

Somente os controles específicos para a tela de consulta serão adicionados neste formulário, lembrando que este formulário será o modelo de todas as telas de consulta.

No formulário altere as seguintes propriedades:

OBJETO		
	Form	
Objeto	**Propriedade**	**Valor**
frmModeloConsulta	Size	520.386
	ShowInTaskBar	True

Insira no formulário os objetos que seguem:

OBJETO		
	GroupBox	
Objeto	**Propriedade**	**Valor**
	Name	grpGrupo
	Location	10,8
	Size	488, 66
	Text	Filtros

O GroupBox é utilizado para agrupamento de outros controles. Dentro do controle GroupBox criado, adicione os seguintes controles:

OBJETO		
	Label	
Objeto	**Propriedade**	**Valor**
	Name	lblDescricao
	Location	6,28 16, 36 28
	Size	74, 16
	Text	Descrição:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtDescricao
	Location	87, 26 34
	Size	297, 23
	Text	

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnPesquisar
	Modifiers	Public
	Location	402, 22 28
	Size	80, 28
	Text	Pesquisar

Voltando ao formulário, adicione os controles que seguem, imediatamente abaixo do GroupBox:

OBJETO		
	DataGridView	
Objeto	Propriedade	Valor
	Name	dgdGrid
	Location	10, 80
	Modifiers	Public
	Size	488, 220

O controle DataGridView possui uma propriedade chamada *Columns* responsável pelas colunas que serão exibidas. Edite a propriedade na barra de propriedades e adicione duas colunas clicando no botão Add. Configure as colunas conforme segue:

PROPRIEDADE		
	Columns	
Código	Propriedade	Valor
	Width	70
	Name	Código
	HeaderText	Código
	DataPropertyName	Código

PROPRIEDADE		
	Columns	
Descrição	Propriedade	Valor
	Width	350
	Name	Descrição
	HeaderText	Descrição
	DataPropertyName	Descrição

Siga adicionando outros controles:

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnNovo
	Location	3, 3 180, 315
	Modifiers	Public
	Size	75, 23
	Text	Novo

OBJETO		
	Button	
Objeto	**Propriedade**	**Valor**
	Name	btnAlterar
	Location	~~84,3~~ 261, 315
	Modifiers	Public
	Size	75, 23
	Text	Alterar

OBJETO		
	Button	
Objeto	**Propriedade**	**Valor**
	Name	btnExcluir
	Location	~~165,3~~ 342, 315
	Modifiers	Public
	Size	75, 23
	Text	Excluir

OBJETO		
	Button	
Objeto	**Propriedade**	**Valor**
	Name	btnSair
	Location	~~246,3~~ 423, 315
	Modifiers	Public
	Size	75, 23
	Text	Sair

A Figura 11.7 ilustra o formulário finalizado.

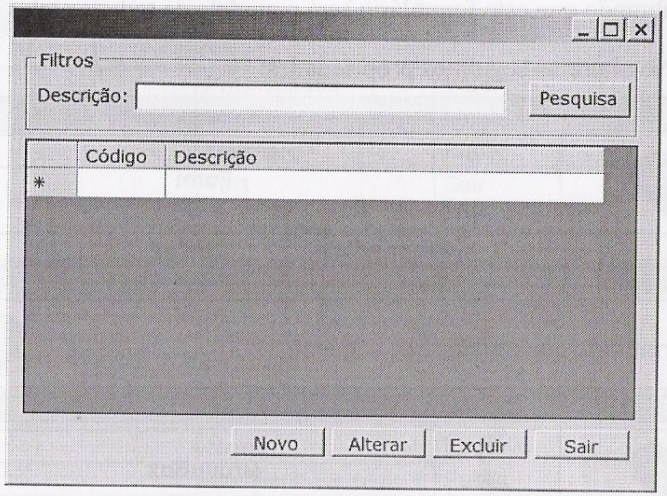


Figura 11.7 – Formulário modelo consulta

Modelo de Cadastro

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Modelos, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmModeloCadastro e clique em OK. Uma nova janela é aberta (Figura 11.8) para a seleção do formulário modelo, selecione e clique em OK.

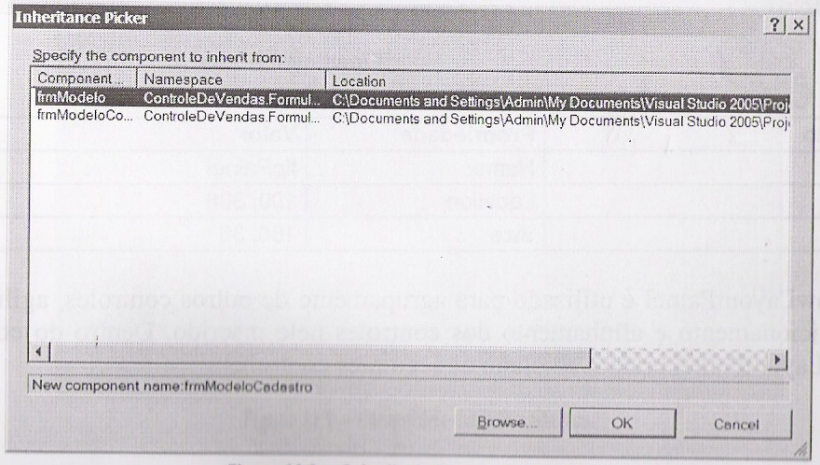


Figura 11.8 – Seleção do formulário modelo.

Somente os controles específicos para a tela de cadastro serão adicionados neste formulário, lembrando que este formulário será o modelo de todas as telas de cadastro.

No formulário altere as seguintes propriedades:

OBJETO		
	Form	
Objeto	Propriedade	Valor
frmModeloCadastro	Size	491, 386

Insira no formulários os objetos que seguem:

OBJETO		
	GroupBox	
Objeto	Propriedade	Valor
	Name	grpCampos
	Location	12, 12
	Size	459, 284
	Modifiers	Public
	Text	

OBJETO		
	FlowLayoutPanel	
Objeto	Propriedade	Valor
	Name	flpPainel
	Location	190, 306
	Size	166, 36

O FlowLayoutPainel é utilizado para agrupamento de outros controles, agilizando o posicionamento e alinhamento dos controles nele inserido. Dentro do controle FlowLayoutPainel criado, adicione os seguintes controles:

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnSair
	Modifiers	Public
	Text	Sair

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnNovo
	Modifiers	Public
	Text	Gravar

A Figura 11.9 ilustra o formulário finalizado.

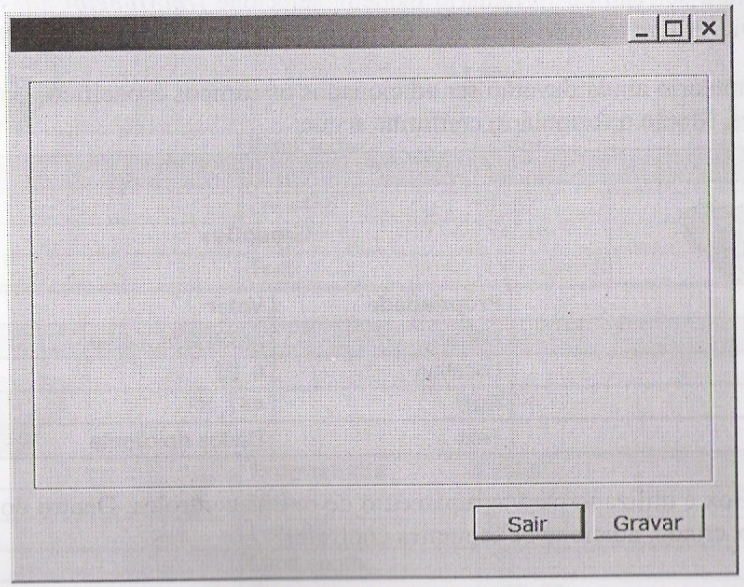


Figura 11.9 – Formulário modelo cadastro.

Formulários de Cadastros

Todos os cadastros do sistema seguem um padrão baseado em duas telas, sendo uma de consulta e outra de cadastro do registro. Quando o usuário clicar no menu do sistema, é aberta uma janela de consulta e a partir dessa consulta são chamadas as rotinas de inclusão, alteração ou exclusão.

Cadastro de Clientes

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Cadastros/Cliente, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome ***frmClienteConsulta*** e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo; selecione ***frmModeloConsulta*** e clique em OK. A interface do formulário já está pronta, pois foram herdados todos os controles do modelo de consulta. Altere a propriedade *text* do formulário para **Consulta de Clientes**.

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Cadastros/Cliente, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome ***frmClienteCadastro*** e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo, selecione ***frmModeloCadastro*** e clique em OK. Altere a propriedade *text* do formulário para **Cadastro de Clientes**.

Neste formulário ainda deverão ser adicionados os campos específicos do cadastro de clientes. Monte o formulário conforme segue:

OBJETO		
	GroupBox	
Objeto	**Propriedade**	**Valor**
	Name	grpDadosCliente
	Location	6, 22
	Size	447, 83
	Text	Dados do cliente

O GroupBox é utilizado para agrupamento de outros controles. Dentro do controle GroupBox criado, adicione os seguintes controles:

OBJETO		
	Label	
Objeto	**Propriedade**	**Valor**
	Name	lblNome
	Location	34, 29
	Size	46, 16
	Text	Nome:

OBJETO		
	TextBox	
Objeto	**Propriedade**	**Valor**
	Name	txtNome
	Location	85, 26
	MaxLength	80
	Size	337, 23

OBJETO		
	Label	
Objeto	**Propriedade**	**Valor**
	Name	lblCpfCnpj
	Location	7, 57
	Size	73, 16
	Text	CPF/CNPJ:

OBJETO		
	TextBox	
Objeto	**Propriedade**	**Valor**
	Name	txtCpfCnpj
	Location	85, 54
	MaxLength	20
	Size	140, 23

Adicione outro GroupBox abaixo do já adicionado para melhor organização do formulário.

OBJETO		
	GroupBox	
Objeto	Propriedade	Valor
	Name	grbEndereco
	Location	6, 111
	Size	447, 167
	Text	Dados do cliente

Dentro do controle GroupBox criado, adicione os seguintes controles:

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblLogradouro
	Location	5, 29
	Size	83, 16
	Text	Logradouro:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtLogradouro
	Location	100, 26
	MaxLength	80
	Size	324, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblBairro
	Location	44, 55
	Size	47, 16
	Text	Bairro:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtBairro
	Location	99, 52
	MaxLength	50
	Size	324, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblCidade
	Location	38, 81
	Size	56, 17
	Text	Cidade:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtCidade
	Location	99, 78
	MaxLength	50
	Size	324, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblUf
	Location	64, 104
	Size	27, 16
	Text	UF:

OBJETO		
	ComboBox	
Objeto	Propriedade	Valor
	DropDownStyle	DropDownList
	Name	cboUf
	Item	SP, RJ, MG, BA
	Location	99, 100
	Size	172, 24

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblCep
	Location	289, 104
	Size	35, 16
	Text	Cep:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtCep
	Location	332, 101
	MaxLength	8
	Size	91, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblEmail
	Location	41, 129
	Size	50, 16
	Text	E-mail:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtEmail
	Location	97, 126
	MaxLength	80
	Size	171, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblTelefones
	Location	17, 131
	Size	74, 16
	Text	Telefones:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtTelefone
	Location	96, 152
	MaxLength	50
	Size	98, 127

A Figura 11.10 ilustra o formulário finalizado.

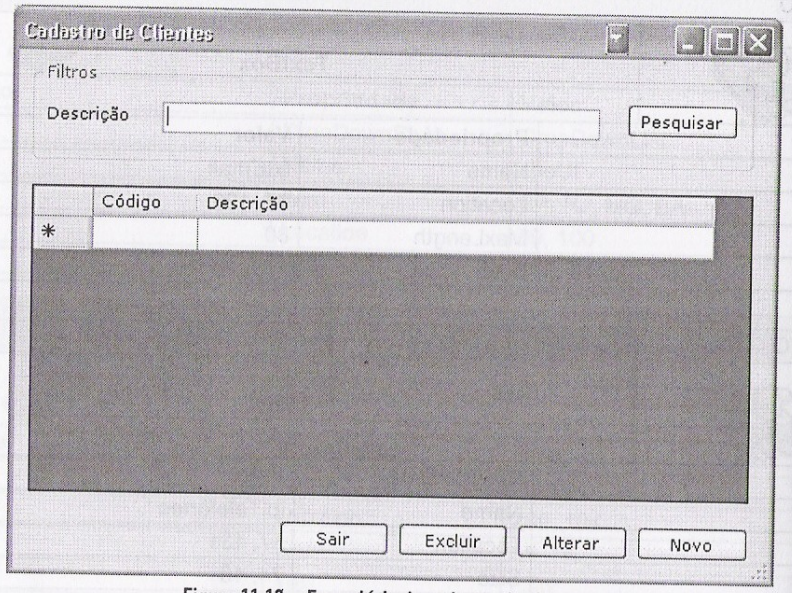


Figura 11.10 – Formulário de cadastro de clientes.

Cadastro de Categorias de Produtos

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Cadastros/Categoria, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmCategoriaConsulta e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo, selecione frmModeloConsulta e clique em OK. A interface do formulário já está pronta, pois foram herdados todos os controles do modelo de consulta. Altere a propriedade *text* do formulário para **Consulta de Categorias**.

Crie agora o formulário de cadastro; no *Solution Explorer* clique com o botão direito do mouse sobre o diretório Cadastros/Categoria, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmCategoriaCadastro e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo, selecione frmModeloCadastro e clique em OK. Altere a propriedade *text* do formulário para **Cadastro de Categorias**.

Neste formulário ainda deverão ser adicionados os campos específicos do cadastro de categorias. Monte o formulário conforme segue:

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblNome
	Location	15, 28
	Size	46, 16
	Text	Nome:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtNome
	Location	66, 25
	MaxLength	80
	Size	337, 23

A Figura 11.11 ilustra o formulário finalizado.

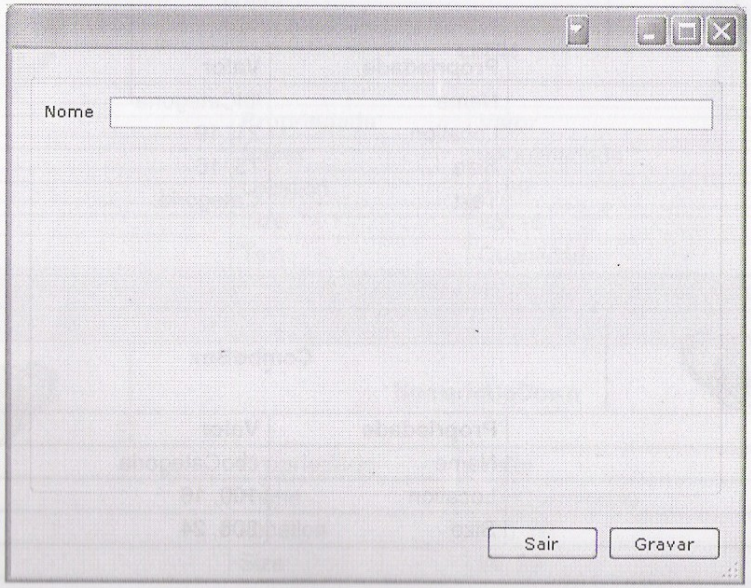


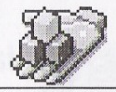
Figura 11.11 – Formulário de cadastro de categorias.

Cadastro de Produtos

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Cadastros/Produto, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmProdutoConsulta e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo; selecione ***frmModeloConsulta*** e clique em OK. A interface do formulário já está pronta, pois foram herdados todos os controles do modelo de consulta. Altere a propriedade *text* do formulário para **Consulta de Produtos**.

Crie agora o formulário de cadastro; no *Solution Explorer* clique com o botão direito do mouse sobre o diretório Cadastros/Produto, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmProdutoCadastro e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo, selecione frmModeloCadastro e clique em OK. Altere a propriedade *text* do formulário para **Cadastro de Produtos**.

Neste formulário ainda deverão ser adicionados os campos específicos do cadastro de produtos. Monte o formulário conforme segue:

OBJETO		
	Label	
Objeto	**Propriedade**	**Valor**
	Name	lblCategoria
	Location	21, 19
	Size	73, 16
	Text	Categoria:

OBJETO		
	ComboBox	
Objeto	**Propriedade**	**Valor**
	Name	cboCategoria
	Location	100, 16
	Size	308, 24

OBJETO		
	Label	
Objeto	**Propriedade**	**Valor**
	Name	lblDescricao
	Location	20, 44
	Size	74, 16
	Text	Descrição:

OBJETO		
	TextBox	
Objeto	**Propriedade**	**Valor**
	Name	txtDescricao
	Location	100, 41
	MaxLength	80
	Size	308, 23

OBJETO		
	Label	
Objeto	**Propriedade**	**Valor**
	Name	lblQuantidade
	Location	9, 69
	Size	85, 16
	Text	Quantidade:

OBJETO		
	NumericUpDown	
Objeto	**Propriedade**	**Valor**
	Name	nudQuantidade
	Location	100, 66
	Size	84, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblValor
	Location	51, 94
	Size	50, 17
	Text	Valor:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtValor
	Location	100, 91
	MaxLength	80
	Size	84, 23

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblAtivo
	Location	50, 119
	Size	44, 16
	Text	Ativo:

OBJETO		
	RadioButton	
Objeto	Propriedade	Valor
	Name	optSim
	Checked	True
	Location	100, 117
	Size	45, 20
	Text	Sim

OBJETO		
	RadioButton	
Objeto	Propriedade	Valor
	Name	optNao
	Checked	False
	Location	151, 117
	Size	47, 20
	Text	Não

A Figura 11.12 ilustra o formulário finalizado.

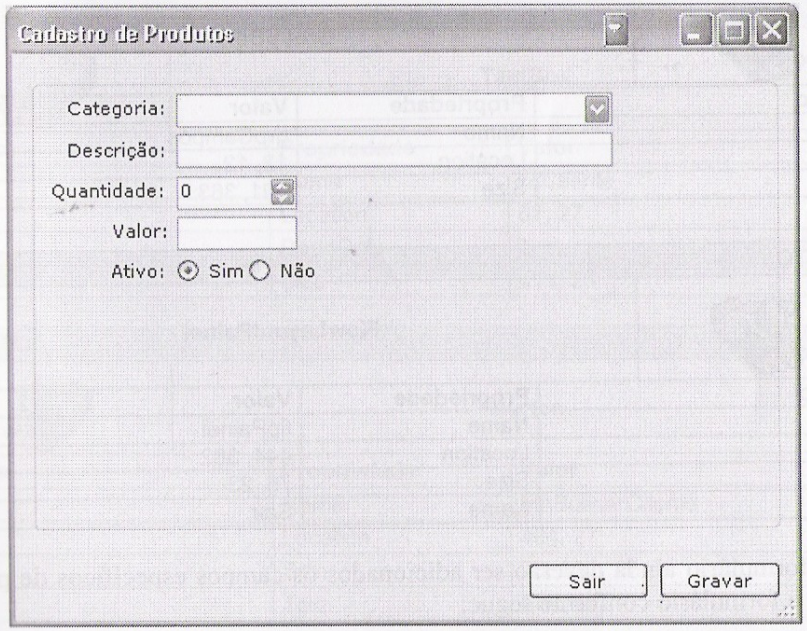


Figura 11.12 – Formulário de cadastro de produtos.

Pedido

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Sistema, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome ***frmPedido*** e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo; selecione frmModeloCadastro e clique em OK.

No formulário altere as seguintes propriedades:

OBJETO		
	Form	
Objeto	Propriedade	Valor
frmPedido	Size	623, 459
	Text	Pedidos

Nos controles herdados do modelo, altere as seguintes propriedades:

OBJETO		
	GroupBox	
Objeto	Propriedade	Valor
	Name	grpCampos
	Location	12, 12
	Size	591, 363

OBJETO		
	FlowLayoutPainel	
Objeto	Propriedade	Valor
	Name	flpPainel
	Location	444, 382
	Size	75, 23
	Name	Sair

Neste formulário ainda deverão ser adicionados os campos específicos de pedido. Monte o formulário conforme segue:

OBJETO		
	GroupBox	
Objeto	Propriedade	Valor
	Name	grbPedido
	Location	6, 22
	Size	579, 126
	Text	Pedido

Dentro do controle GroupBox criado, adicione os seguintes controles:

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblCliente
	Location	7, 30
	Size	55, 16
	Text	Cliente:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtCliente
	Location	67, 27
	ReadOnly	True
	Size	411, 23

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnBuscarCliente
	Location	484, 27
	Size	75, 23
	Text	Buscar

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblData
	Location	21, 61
	Size	41, 16
	Text	Data:

OBJETO		
	DateTime	
Objeto	Propriedade	Valor
	Name	dtpData
	CustomFormat	dd/MM/yyyy
	Format	Custom
	Location	67, 58
	Size	112, 23

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnAdicionar
	Location	6, 93
	Size	90, 23
	Text	Adicionar

Adicione outro GroupBox abaixo do já existente e adicione os seguintes controles:

OBJETO		
	GroupBox	
Objeto	Propriedade	Valor
	Name	grbItem
	Location	6, 154
	Size	579, 199
	Text	Item

Dentro do controle GroupBox criado, adicione os seguintes controles:

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblProduto
	Location	6, 25
	Size	61, 16
	Text	Produto:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtProduto
	Location	75, 22
	ReadOnly	True
	Size	403, 23

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnBuscarProduto
	Location	484, 22
	Size	75, 23
	Text	Buscar

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblQuantidade
	Location	7, 58
	Size	58, 16
	Text	Qtdade:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtQuantidade
	Location	75, 55
	MaxLength	5
	Size	100, 23
	Texto	

OBJETO		
	Label	
Objeto	Propriedade	Valor
	Name	lblValor
	Location	183, 58
	Size	43, 16
	Text	Valor:

OBJETO		
	TextBox	
Objeto	Propriedade	Valor
	Name	txtValor
	Location	226, 55
	MaxLength	10
	Size	100, 23
	Texto	

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnAdicionarItem
	Location	333, 55
	Size	83, 23
	Text	Adicionar

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnRemoverItem
	Location	421, 55
	Size	75, 23
	Text	Remover

OBJETO		
	DataGridView	
Objeto	Propriedade	Valor
	Name	dgvItem
	Location	10, 84
	Size	563, 109

A Figura 11.13 ilustra o formulário finalizado.

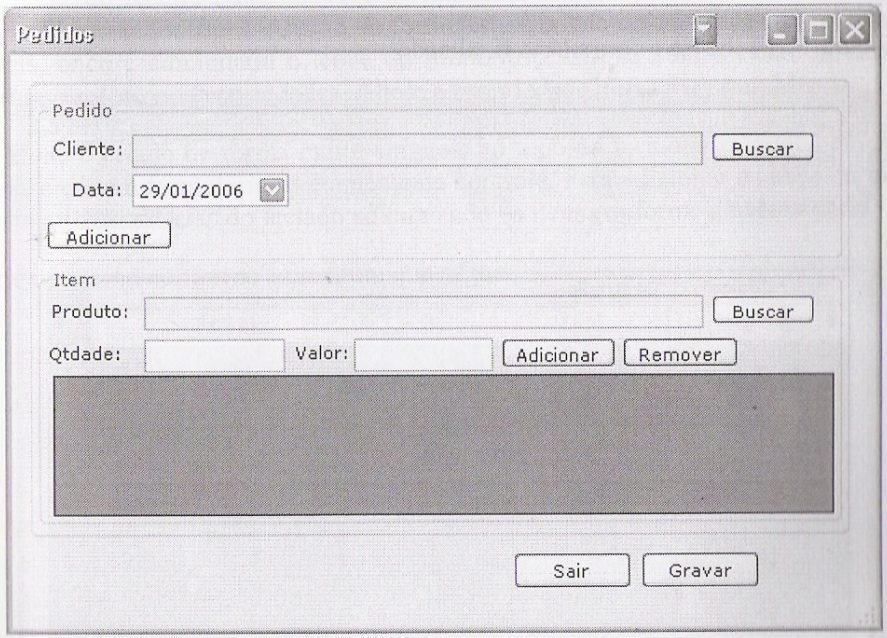


Figura 11.13. Formulário de pedidos.

O formulário de pedidos possui dois campos, cliente e produto, que necessitam de um outro formulário de busca para facilitar a seleção. Será criada uma tela padrão de consulta para tratar os dois casos.

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Sistema, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmConsulta e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo; selecione frmModeloConsulta e clique em OK.

Insira o controle que segue no formulário:

OBJETO		
	Button	
Objeto	Propriedade	Valor
	Name	btnSelecionar
	Location	10; 317
	Size	83, 23
	Text	Selecionar

A Figura 11.14 ilustra o formulário finalizado.

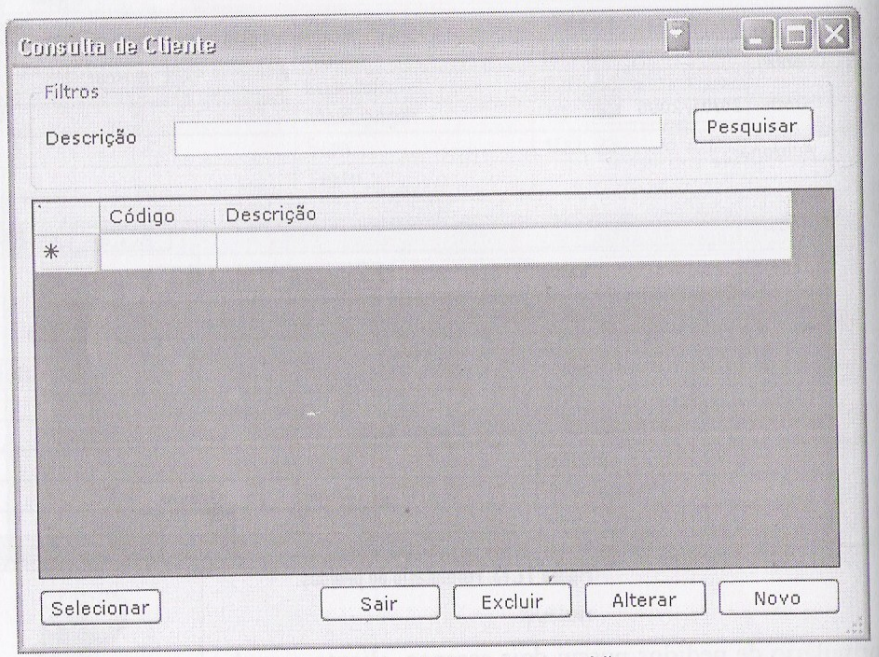


Figura 11.14. Formulário de consulta padrão.

Menu Principal

No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Sistema, Add e Add New Item. Selecione o template *Inherited Form* (formulário herdado) e o nome frmMenu e clique em OK. Uma nova janela é aberta para a seleção do formulário modelo; selecione frmModelo e clique em OK.

No formulário altere as seguintes propriedades:

OBJETO		
	Form	
Objeto	Propriedade	Valor
	Text	Menu Principal – Sistema de Vendas
	Size	886, 639

No menu do sistema adicione um controle MenuStrip, localizado na *ToolBox*, categoria *Menus* e *ToolBars*. Abaixo do formulário é criada uma área de controles invisíveis, onde é adicionado o ícone do MenuStrip. Clique com o botão direito do mouse e selecione Propriedades. Altere a propriedade *name* para mnuMenu.

O Menu é criado de forma muito simples: no topo do formulário, clique em *Type Here* e edite o formulário diretamente no controle. Para adicionar os itens no menu basta utilizar as setas do teclado adicionando os itens conforme a necessidade.

Crie um menu conforme a estrutura que segue:

- Sistema
  - Sobre
  - Sair
- Cadastros
  - Clientes
  - Produtos
    - Produto
    - Categoria
- Vendas

A Figura 11.15 ilustra o menu do sistema.

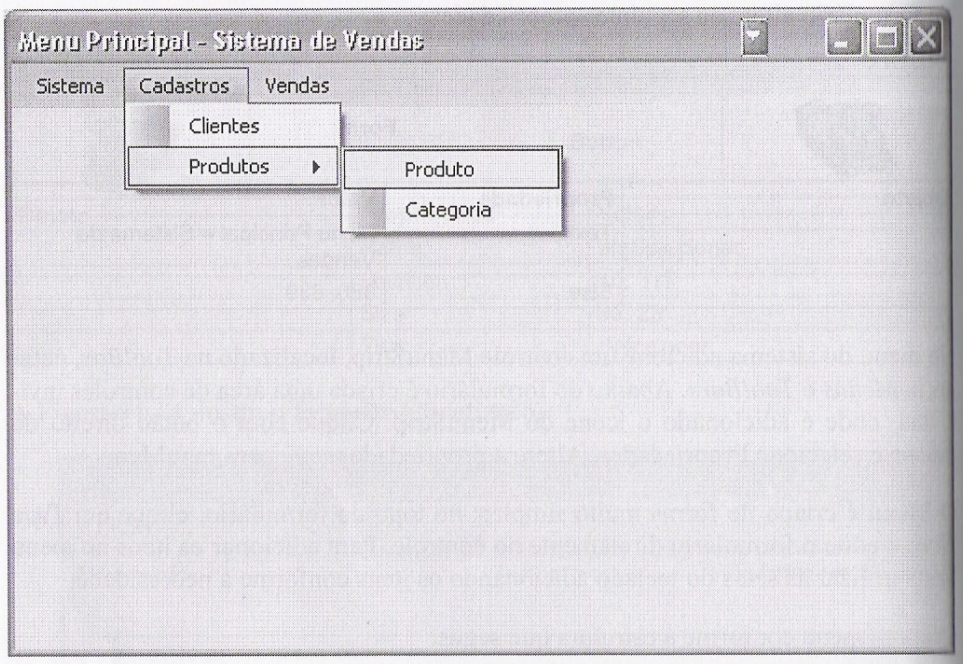


Figura 11.15 – Menu do Sistema.

Para definir o formulário como menu principal do sistema, na classe *Program.cs* altere o `Application.Run` para `Application.Run(new Formulários.Sistema.frmMenu());`. Com isso o formulário de menu passa a ser o formulário inicial do sistema.

Regra de negócio

A arquitetura do projeto em questão é moderna e flexível, isolando a regra de negócio da interface do sistema, facilitando a manutenção conforme a mudança de requisitos.

O termo "Regra de Negócios" no contexto de projetos de sistemas pode ser definido como um conjunto de regras que executam as tarefas do projeto. Portanto, todas as operações (inclusão, alteração, exclusão, consulta de dados) realizadas no sistema são organizadas em classes e codificadas na "Regra de Negócio".

Dentro do mesmo projeto, no menu clique em File, Add e New Project. Selecione o Template **Class Library, name DllControleDeVendas** e clique em OK.

O projeto é criado dentro da *Solution,* onde será iniciada a construção da regra de negócio. Abra o *Solution Explorer* (Ctrl + Alt + L), clique com o botão direito do mouse sobre a Class1 e exclua a classe.

Organização do projeto

Crie toda a estrutura do sistema, abra o *Solution Explorer* clique com o botão direito do mouse sobre o projeto, Add, New Folder. A criação é semelhante ao Windows Explorer. Para alterar os nomes de diretórios basta selecionar o diretório com o botão direito do mouse e clicar em Rename. Crie toda a estrutura conforme a Figura 11.16.

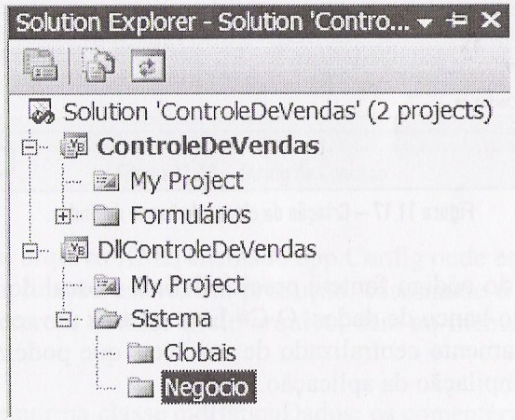


Figura 11.16 – Estrutura de diretórios.

Classes Globais

Todo o suporte de que as regras de negócio necessitam são tratadas dentro de classes adicionadas no diretório Globais, tarefas como banco de dados, variáveis públicas etc.

Será criada uma classe chamada cldBancoDados, onde todas as operações de banco de dados devem ser tratadas. No *Solution Explorer* clique com o botão direito do mouse sobre o diretório Globais, Add e Class. Uma nova janela é aberta (Figura 11.17) para a seleção do template (Class) e o nome, cldBancoDados. Clique em OK.

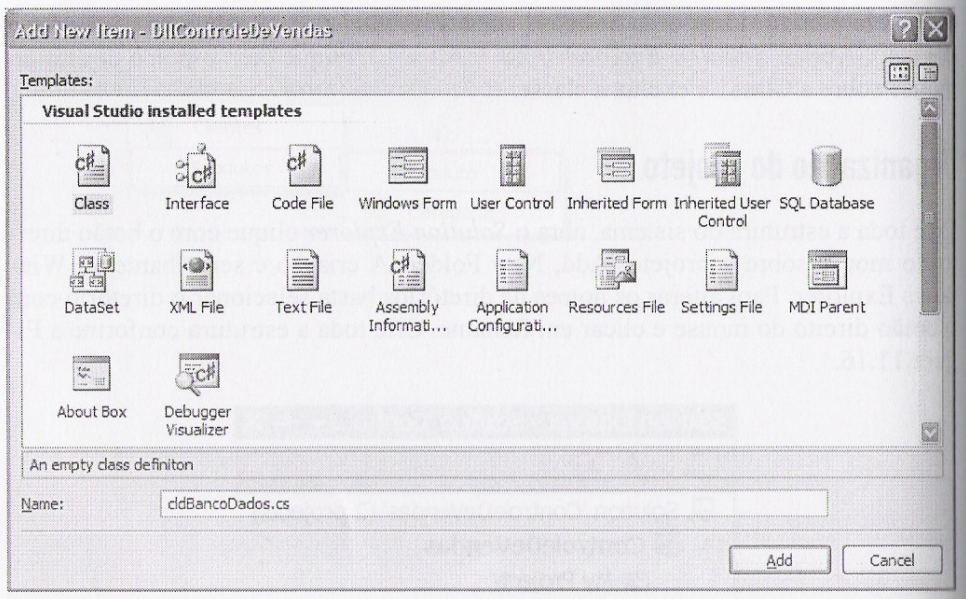


Figura 11.17 – Criação da classe de banco de dados.

Antes da digitação do código fonte é preciso definir o local do armazenamento da string de conexão do banco de dados. O C# Express traz um recurso muito interessante para armazenamento centralizado de variáveis que podem ser alteradas sem que exija uma recompilação da aplicação.

No *Solution Explorer* dê um duplo clique em *My Project*. Selecione a aba *Settings*. Uma listagem é aberta. Adicione uma nova linha conforme tabela a seguir:

Name	Type	Scope	Caminho
Conexao	ConnectionString	Application	Caminho selecionado

Tabela 11.1 – Valores da string de conexao.

O caminho do banco de dados deve ser selecionado conforme o caminho já criado no início do projeto (Figura 11.18).

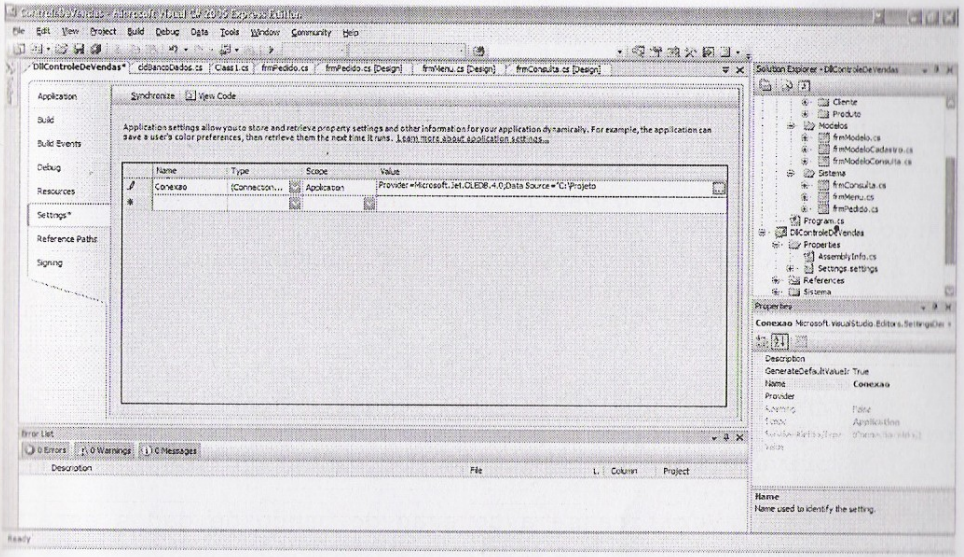


Figura 11.18 – String de Conexao.

A aplicação gera um arquivo XML chamado app.Config onde esse valor fica armazenado. Quando a aplicação estiver em produção, caso mude o caminho do banco de dados, basta alterarem o valor deste arquivo, sem ao menos a necessidade de fechar a aplicação.

Digite o código a seguir na classe cldBancoDados; os comentários explicam o funcionamento:

```
// Adiciona as Namespaces necessárias nesta classe
using System;
using System.Data.OleDb;
using System.Data;
using System.Configuration;

namespace DllControleDeVendas.Sistema.Globais
{
    public class cldBancoDados
    {
        // Passa para variável StringConexao o valor do
arquivo de configuração

        //string StringConexao = (string)["Conexao"];
        string StringConexao =
Properties.Settings.Default.Conexao;
```

```
        // Função utilizada para abrir a conexão com o banco
de dados
        private OleDbConnection AbreBanco()
        {
            OleDbConnection Conn = new OleDbConnection();
            // Passa o valor da String de Conexão e abre o
banco de dados
            Conn.ConnectionString = StringConexao;
            Conn.Open();
            return Conn;
        }

        //  Rotina utilizada para fechar a conexão com o
banco de dados
        private void FechaBanco(OleDbConnection Conn)
        {
            // Verifica se a conexão do banco de dados está
aberta
            if (Conn.State == ConnectionState.Open)
            {
                Conn.Close();
            }
        }

        // Função utilizada para executar comandos no banco
de dados
        public void ExecutaComando(string strQuery)
        {
            OleDbConnection Conn = new OleDbConnection();
            // Cria o objeto de conexão
            try
            {
                // Abre o banco da dados
                Conn = AbreBanco();

                // Cria o objeto de comando
                OleDbCommand cmdComando = new OleDbCommand();

                // Passa os valores da Query SQL, tipo do
comando, conexão e executa o comando
                cmdComando.CommandText = strQuery;
                cmdComando.CommandType = CommandType.Text;
                cmdComando.Connection = Conn;
                cmdComando.ExecuteNonQuery();
```



```
            }
            // Tratamento de excessões
            catch (Exception ex)
            {
                throw ex;
            }
            finally
            {
                // Em caso de erro ou não, o finally é
executado para fechar a conexão com  o banco de dados
                FechaBanco(Conn);
            }
        }

        // Função utilizada para executar comandos com algum
retorno do tipo Integer
        public int ExecutaComandoRetorno(string strQuery)
        {
            // Cria o objeto de conexão
            OleDbConnection Conn = new OleDbConnection();

            // Declara um novo DataReader
            OleDbDataReader dr;
            try
            {
                // Abre a conexão com o banco de dados
                Conn = AbreBanco();

                // Cria o objeto de comando
                OleDbCommand cmdComando = new OleDbCommand();

                // Passa os valores da Query SQL, tipo do
comando, conexão e executa o comando
                cmdComando.CommandText = strQuery;
                cmdComando.CommandType = CommandType.Text;
                cmdComando.Connection = Conn;
                cmdComando.ExecuteNonQuery();

                // Cria uma nova Query, que busca o valor do
Identity gerado pelo banco de dados
                cmdComando.CommandText = "Select
@@Identity;";

                // Executa a leitura do comando
                dr = cmdComando.ExecuteReader();
```

```
                // Lê o DataReader
                dr.Read();

                // Retorna o valor
                return (int)dr[0];
            }
            // Tratamento de exceções
            catch (Exception ex)
            {
                throw ex;
            }
            finally
            {
            // Em caso de erro ou não, o finally é executado
para fechar a conexão com  o banco de dados
                FechaBanco(Conn);
            }
        }
        // Função utilizada para retornar um DataSet a partir
de uma Query Sql
        public DataSet RetornaDataSet(string strQuery)
        {
            // Cria o objeto de conexão
            OleDbConnection Conn = new OleDbConnection();
            try
            {
                // Abre a conexão com o banco de dados
                Conn = AbreBanco();


                // Cria o objeto de comando
                OleDbCommand cmdComando = new OleDbCommand();

                // Passa os valores da Query SQL, tipo do
comando, conexão e executa o comando
                cmdComando.CommandText = strQuery;
                cmdComando.CommandType = CommandType.Text;
                cmdComando.Connection = Conn;

                // Declara um DataAdapter
                OleDbDataAdapter daAdaptador = new
OleDbDataAdapter();

                // Declara um DataSet
                DataSet dsDataSet = new DataSet();
```



```
                // Passa o comando a ser executado pelo
DataAdapter
                daAdaptador.SelectCommand = cmdComando;

                // O DataAdapter faz o conexão com o banco de
dados, carrega o DataSet e fecha a conexão
                daAdaptador.Fill(dsDataSet);

                // Retorna o DataSet carregado
                return dsDataSet;
            }
            // tratamento de exceções
            catch (Exception ex)
            {
                throw ex;
            }
            finally
            {
            // Em caso de erro ou não, o finally é executado
para fechar a conexão com  o banco de dados
              FechaBanco(Conn);
            }
        }
        // Função utilizada para retornar um DataReader a
partir de uma Query Sql
        public OleDbDataReader RetornaDataReader(string
strQuery)
        {
            // Cria o objeto de conexão
            OleDbConnection Conn = new OleDbConnection();
            try
            {
                Conn = AbreBanco();
                OleDbCommand cmdComando = new OleDbCommand();
                cmdComando.CommandText = strQuery;
                cmdComando.CommandType = CommandType.Text;
                cmdComando.Connection = Conn;
                return cmdComando.ExecuteReader();
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
    }
}
```

A classe de banco de dados efetua todas as tarefas pertinentes a banco de dados, e todos os métodos desta classe centralizam todas as funcionalidades.

A classe clnFuncoesGerais possui um Enumerador para operações dos formulários.

```
namespace DllControleDeVendas.Sistema.Globais
{
    public class clnFuncoesGerais
    {
        public enum Operacao
        {
            Inclusao,
            Alteracao
        }
    }
}
```

Classes de Negócio

Implementar as classes de negócios é uma tarefa simples, já que temos a classe que trata toda a informação proveniente do banco de dados desenvolvida.

Classe de Categoria, clnCategoria

```
// Adiciona as Namespaces necessárias nesta classe
using System.Data;
using System.Data.OleDb;
using System.Text;
using DllControleDeVendas.Sistema.Globais;

namespace DllControleDeVendas.Sistema.Negocio
{
    public class clnCategoria
    {
        //Cria todos os métodos internos e propriedades
externas com os mesmos atributos do banco de dados

        private int _cat_ID;
        public int cat_ID
        {
            get
            {
                return _cat_ID;
            }
```



```
            set
            {
                _cat_ID = value;
            }
        }
        private string _cat_Descricao;
        public string cat_Descricao
        {
            get
            {
                return _cat_Descricao;
            }
            set
            {
                _cat_Descricao = value;
            }
        }
        // Função utilizada para retornar um DataSet com o
select indicado

        public DataSet Listar(string strDescricao)
        {
            // Cria um StringBuilder para concatenar a Query
Sql

            StringBuilder strQuery = new StringBuilder();
            strQuery.Append(" SELECT cat_Id as Codigo,
cat_Descricao as Descricao ");
            strQuery.Append(" FROM Categoria ");

            // Adiciona o filtro do parâmetro
            if (!(strDescricao.Equals(string.Empty)))
            {
                strQuery.Append(" WHERE cat_Descricao like
'%" + strDescricao + "%' ");
            }
            // Executa o método RetornaDataSet da classe de
banco de dados e retorna o DataSet
            cldBancoDados cldBancoDados = new
cldBancoDados();
            return
cldBancoDados.RetornaDataSet(strQuery.ToString());
        }

        public void Excluir(int intCodigo)
        {
```

```
            // Cria um StringBuilder para concatenar a Query
Sql
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" DELETE FROM categoria ");
            strQuery.Append(" WHERE cat_id = " + intCodigo +
"");

            // Executa o método ExecutaComando da classe de
banco de dados
            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public void Gravar()
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" INSERT INTO categoria ");
            strQuery.Append(" ( ");
            strQuery.Append(" cat_Descricao ");
            strQuery.Append(" ) ");
            strQuery.Append(" VALUES ( ");
            strQuery.Append(" '" + cat_Descricao + "' ");
            strQuery.Append(" ); ");

            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public void Alterar()
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" UPDATE categoria ");
            strQuery.Append(" SET ");
            strQuery.Append(" cat_Descricao = '" +
_cat_Descricao + "' ");
            strQuery.Append(" WHERE ");
            strQuery.Append(" cat_ID = " + _cat_ID + " ");
```



```
            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public OleDbDataReader ListarCategoria(int intCodigo)
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" SELECT cat_Descricao ");
            strQuery.Append(" FROM categoria ");
            strQuery.Append(" WHERE cat_id = " + intCodigo +
" ");
            cldBancoDados cldBancoDados = new
cldBancoDados();

            return
cldBancoDados.RetornaDataReader(strQuery.ToString());
        }
    }
}
```

Classe de Produto, clnProduto

```
using System.Data;
using System.Data.OleDb;
using System.Text;
using DllControleDeVendas.Sistema.Globais;

namespace DllControleDeVendas.Sistema.Negocio
{
    public class clnProduto
    {
        // Cria todos os métodos internos e propriedades
externas com os mesmos atributos do banco de dados
        private int _pro_id;

        public int pro_id
        {
            get
            {
                return _pro_id;
            }
            set
```

```
        {
            _pro_id = value;
        }
    }
    private clnCategoria _clnCategoria;

    public clnCategoria clnCategoria
    {
        get
        {
            return _clnCategoria;
        }
        set
        {
            _clnCategoria = value;
        }
    }
    private string _pro_Descricao;

    public string pro_Descricao
    {
        get
        {
            return _pro_Descricao;
        }
        set
        {
            _pro_Descricao = value;
        }
    }
    private int _pro_QtdeEstoque;

    public int pro_QtdeEstoque
    {
        get
        {
            return _pro_QtdeEstoque;
        }
        set
        {
            _pro_QtdeEstoque = value;
        }
    }
    private decimal _pro_Valor;

    public decimal pro_Valor
```



```
    {
        get
        {
            return _pro_Valor;
        }
        set
        {
            _pro_Valor = value;
        }
    }
    private bool _pro_Ativo;

    public bool pro_Ativo
    {
        get
        {
            return _pro_Ativo;
        }
        set
        {
            _pro_Ativo = value;
        }
    }

    public void Gravar()
    {
        // Cria um StringBuilder para concatenar a Query
Sql
        StringBuilder strQuery = new StringBuilder();

        strQuery.Append(" INSERT INTO produto ");
        strQuery.Append(" ( ");
        strQuery.Append(" cat_Id ");
        strQuery.Append(", pro_Descricao ");
        strQuery.Append(", pro_QtdeEstoque ");
        strQuery.Append(", pro_Valor ");
        strQuery.Append(", pro_Ativo ");
        strQuery.Append(" ) ");
        strQuery.Append(" VALUES ( ");
        strQuery.Append(" '" + _clnCategoria.cat_ID + "'
");
        strQuery.Append(", '" + _pro_Descricao + "' ");
        strQuery.Append(", '" + _pro_QtdeEstoque + "' ");
        strQuery.Append(", '" + _pro_Valor + "' ");
        strQuery.Append(", " + _pro_Ativo + " ");
        strQuery.Append(" ); ");
```

```
            // Executa o método ExecutaComando da classe de
banco de dados
            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public DataSet Listar(string strDescricao)
        {
            // Cria um StringBuilder para concatenar a Query
Sql
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" SELECT pro_Id as Codigo,
pro_Descricao as Descricao ");
            strQuery.Append(" FROM produto ");

            // Adiciona o filtro do parametro
            if (!(strDescricao.Equals(string.Empty)))
            {
                strQuery.Append(" WHERE pro_Descricao like
'%" + strDescricao + "%' ");
            }

            // Executa o método RetornaDataSet da classe de
banco de dados e retorna o DataSet
            cldBancoDados cldBancoDados = new
cldBancoDados();
            return
cldBancoDados.RetornaDataSet(strQuery.ToString());
        }

        public void Excluir(int intCodigo)
        {
            // Cria um StringBuilder para concatenar a Query
Sql
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" DELETE FROM produto ");
            strQuery.Append(" WHERE pro_id = " + intCodigo +
"");

            // Executa o método ExecutaComando da classe de
banco de dados
```



```
            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public void Alterar()
        {
            // Cria um StringBuilder para concatenar a Query
Sql
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" UPDATE produto ");
            strQuery.Append(" SET ");
            strQuery.Append(" cat_Id = '" +
_clnCategoria.cat_ID + "' ");
            strQuery.Append(", pro_Descricao = '" +
_pro_Descricao + "' ");
            strQuery.Append(", pro_QtdeEstoque = '" +
_pro_QtdeEstoque + "' ");
            strQuery.Append(", pro_Valor = '" + _pro_Valor +
"' ");
            strQuery.Append(", pro_Ativo = " + _pro_Ativo +
");
            strQuery.Append(" WHERE ");
            strQuery.Append(" pro_ID = " + _pro_id + " ");

            // Executa o método ExecutaComando da classe de
banco de dados
            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public OleDbDataReader ListarProduto(int intCodigo)
        {
            // Cria um StringBuilder para concatenar a Query
Sql
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" SELECT cat_Id, pro_Descricao,
pro_QtdeEstoque, pro_Valor, pro_Ativo ");
            strQuery.Append(" FROM produto ");
            strQuery.Append(" WHERE pro_id = " + intCodigo +
" ");
```

```
            // Executa o método RetornaDataReader da classe
de banco de dados e retorna o DataReader
            cldBancoDados cldBancoDados = new
cldBancoDados();
            return
cldBancoDados.RetornaDataReader(strQuery.ToString());
        }
    }
}
```

Classe de Cliente, clnCliente

Todas as funcionalidades são idênticas às de outros cadastros. Como o acesso ao banco de dados é feito em uma classe centralizadora, o código fica sempre praticamente idêntico, mudando apenas o comando SQL que será executado.

```
using System.Data;
using System.Data.OleDb;
using System.Text;
using System;
using DllControleDeVendas.Sistema.Globais;

namespace DllControleDeVendas.Sistema.Negocio
{
    public class clnCliente
    {
        private int _cli_Id;

        public int cli_Id
        {
            get
            {
                return _cli_Id;
            }
            set
            {
                _cli_Id = value;
            }
        }
        private string _cli_NomeRazao;

        public string cli_NomeRazao
        {
            get
            {
```



```
                return _cli_NomeRazao;
            }
            set
            {
                _cli_NomeRazao = value;
            }
        }
        private string _cli_CNPJCPF;

        public string cli_CNPJCPF
        {
            get
            {
                return _cli_CNPJCPF;
            }
            set
            {
                _cli_CNPJCPF = value;
            }
        }
        private string _cli_Logradouro;

        public string cli_Logradouro
        {
            get
            {
                return _cli_Logradouro;
            }
            set
            {
                _cli_Logradouro = value;
            }
        }
        private string _cli_Bairro;

        public string cli_Bairro
        {
            get
            {
                return _cli_Bairro;
            }
            set
            {
                _cli_Bairro = value;
            }
        }
```

```
    private string _cli_Cidade;

    public string cli_Cidade
    {
        get
        {
            return _cli_Cidade;
        }
        set
        {
            _cli_Cidade = value;
        }
    }
    private string _cli_Uf;

    public string cli_Uf
    {
        get
        {
            return _cli_Uf;
        }
        set
        {
            _cli_Uf = value;
        }
    }
    private string _cli_CEP;

    public string cli_CEP
    {
        get
        {
            return _cli_CEP;
        }
        set
        {
            _cli_CEP = value;
        }
    }
    private string _cli_Email;

    public string cli_Email
    {
        get
        {
            return _cli_Email;
```



```
        }
        set
        {
            _cli_Email = value;
        }
    }
    private string _cli_Fones;

    public string cli_Fones
    {
        get
        {
            return _cli_Fones;
        }
        set
        {
            _cli_Fones = value;
        }
    }
    private DateTime _cli_DtCadastro;

    public DateTime cli_DtCadastro
    {
        get
        {
            return _cli_DtCadastro;
        }
        set
        {
            _cli_DtCadastro = value;
        }
    }

    public void Gravar()
    {
        StringBuilder strQuery = new StringBuilder();

        strQuery.Append(" INSERT INTO cliente ");
        strQuery.Append(" ( ");
        strQuery.Append(" cli_NomeRazao ");
        strQuery.Append(", cli_CNPJCPF ");
        strQuery.Append(", cli_Logradouro ");
        strQuery.Append(", cli_Bairro ");
        strQuery.Append(", cli_Cidade ");
        strQuery.Append(", cli_Uf ");
        strQuery.Append(", cli_CEP ");
```

```
            strQuery.Append(", cli_Email ");
            strQuery.Append(", cli_Fones ");
            strQuery.Append(", cli_DtCadastro ");
            strQuery.Append(" ) ");
            strQuery.Append(" VALUES ( ");
            strQuery.Append(" '" + cli_NomeRazao + "' ");
            strQuery.Append(", '" + _cli_CNPJCPF + "' ");
            strQuery.Append(", '" + _cli_Logradouro + "' ");
            strQuery.Append(", '" + _cli_Bairro + "' ");
            strQuery.Append(", '" + _cli_Cidade + "' ");
            strQuery.Append(", '" + _cli_Uf + "' ");
            strQuery.Append(", '" + _cli_CEP + "' ");
            strQuery.Append(", '" + _cli_Email + "' ");
            strQuery.Append(", '" + _cli_Fones + "' ");
            strQuery.Append(", '" + _cli_DtCadastro + "' ");
            strQuery.Append(" ); ");
            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public void Alterar()
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" UPDATE cliente ");
            strQuery.Append(" SET ");
            strQuery.Append(" cli_NomeRazao = '" +
_cli_NomeRazao + "' ");
            strQuery.Append(", cli_CNPJCPF = '" +
_cli_CNPJCPF + "' ");
            strQuery.Append(", cli_Logradouro = '" +
_cli_Logradouro + "' ");
            strQuery.Append(", cli_Bairro = '" + _cli_Bairro
+ "' ");

            strQuery.Append(", cli_Cidade = '" + _cli_Cidade
+ "' ");

            strQuery.Append(", cli_Uf = '" + _cli_Uf + "' ");
            strQuery.Append(", cli_CEP = '" + _cli_CEP + "'
");

            strQuery.Append(", cli_Email = '" + _cli_Email +
"' ");

            strQuery.Append(", cli_Fones = '" + _cli_Fones +
"' ");

            strQuery.Append(" WHERE ");
```



```
            strQuery.Append(" cli_ID = " + _cli_Id + " ");

            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public DataSet Listar(string strDescricao)
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" SELECT cli_Id as Codigo,
cli_NomeRazao as Descricao ");
            strQuery.Append(" FROM cliente ");

            if (!(strDescricao.Equals(string.Empty)))
            {
                strQuery.Append(" WHERE cli_NomeRazao like
'%" + strDescricao + "%' ");
            }

            cldBancoDados cldBancoDados = new
cldBancoDados();
            return
cldBancoDados.RetornaDataSet(strQuery.ToString());
        }

        public void Excluir(int intCodigo)
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" DELETE FROM cliente ");
            strQuery.Append(" WHERE cli_id = " + intCodigo +
"");

            cldBancoDados cldBancoDados = new
cldBancoDados();

cldBancoDados.ExecutaComando(strQuery.ToString());
        }

        public OleDbDataReader ListarCliente(int intCodigo)
        {

            StringBuilder strQuery = new StringBuilder();
```

```
            strQuery.Append(" SELECT cli_NomeRazao,
cli_CNPJCPF, cli_Logradouro, cli_Bairro, cli_Cidade, cli_Uf,
cli_CEP, cli_Email, cli_Fones ");
            strQuery.Append(" FROM cliente ");
            strQuery.Append(" WHERE cli_id = " + intCodigo +
" ");

            cldBancoDados cldBancoDados = new
cldBancoDados();
            return
cldBancoDados.RetornaDataReader(strQuery.ToString());
        }
    }
}
```

Classe de Pedido, clnPedido

```
using System;
using System.Text;
using System.Data;
using DllControleDeVendas.Sistema.Globais;

namespace DllControleDeVendas.Sistema.Negocio
{
    public class clnPedido
    {
        private int _ped_id;
        public int ped_id
        {
            get
            {
                return _ped_id;
            }
            set
            {
                _ped_id = value;
            }
        }
        private clnCliente _clnCliente;
        public clnCliente clnCliente
        {
            get
            {
                return _clnCliente;
            }
```



```
            set
            {
                _clnCliente = value;
            }
        }
        private DateTime _ped_DtEncomenda;
        public DateTime ped_DtEncomenta
        {
            get
            {
                return _ped_DtEncomenda;
            }
            set
            {
                _ped_DtEncomenda = value;
            }
        }
        private decimal _ped_Valor;
        public decimal ped_Valor
        {
            get
            {
                return _ped_Valor;
            }
            set
            {
                _ped_Valor = value;
            }
        }
        private char _ped_Status;
        public char ped_Status
        {
            get
            {
                return _ped_Status;
            }
            set
            {
                _ped_Status = value;
            }
        }
        public int Gravar()
        {
            StringBuilder strQuery = new StringBuilder();

            strQuery.Append(" INSERT INTO pedido ");
```

```
            strQuery.Append(" ( ");
            strQuery.Append(" cli_Id ");
            strQuery.Append(", ped_DtEncomenda ");
            strQuery.Append(", ped_Valor ");
            strQuery.Append(", ped_Status ");
            strQuery.Append(" ) ");
            strQuery.Append(" VALUES ( ");
            strQuery.Append(" '" + _clnCliente.cli_Id + "'
");
            strQuery.Append(", '" + _ped_DtEncomenda + "' ");
            strQuery.Append(", '" + _ped_Valor + "' ");
            strQuery.Append(", '" + _ped_Status + "' ");
            strQuery.Append(" ); ");

            cldBancoDados cldBancoDados = new
cldBancoDados();
            return
cldBancoDados.ExecutaComandoRetorno(strQuery.ToString());
        }
    }
}
```

Integrando Interface e Regra de Negócio

Após as duas primeiras etapas da aplicação, o próximo passo é integrar as regras de negócio com a interface gráfica. Para isso é necessário "informar" à interface gráfica referenciando a regra de negócio no projeto. Clique com o botão direito do mouse sobre o projeto ***ControleDeVendas*** e *Add Reference*.

A janela de referência (Figura 11.19) é aberta; como estamos trabalhando com os dois projetos dentro de uma mesma solução, selecione a aba Projects e dê um duplo clique no projeto DllControleDeVendas. A referência já foi adicionada.

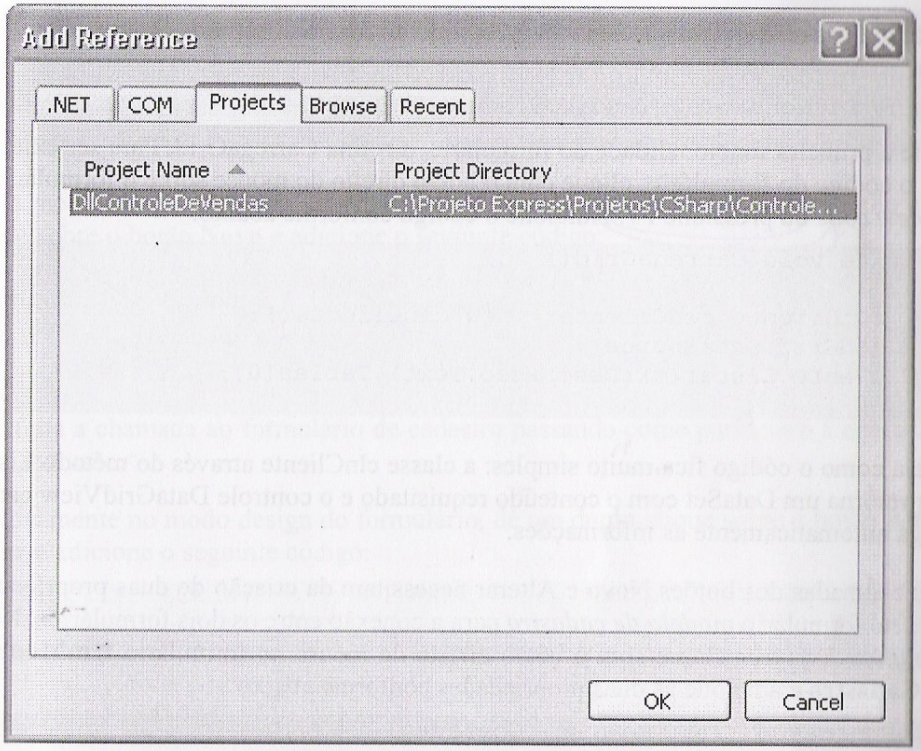


Figura 11.19 – Janela de referência.

Cadastro de Clientes

O cadastro de clientes possui dois formulários: consulta e cadastro. O primeiro a ser implementado será o de consulta, que será responsável pela chamada do formulário de cadastro.

Formulário de Consulta

No *Solution Explorer* dê um duplo clique no formulário **frmClienteConsulta** e o modo design do formulário é aberto. O funcionamento da tela é muito simples, o botão pesquisar preenche o controle DataGridView conforme um filtro digitado ou não. A partir dos botões de ação Novo e Alterar, o formulário de cadastro é chamado. No caso de alteração e exclusão é utilizado o produto selecionado no DataGridView, e o botão sair fecha o formulário.

Adicione as referências que seguem no código do formulário (*view code ou F7*):

```
using DllControleDeVendas.Sistema.Globais;
using DllControleDeVendas.Sistema.Negocio;
```

Crie a primeira funcionalidade do formulário, a rotina CarregaGrid. Para ser exibido o código do formulário, clique com o botão direito do mouse sobre o formulário e *view code* ou pressione F7. Adicione o código conforme abaixo:

```
private void CarregaGrid()
 {
    clnCliente cldCliente = new clnCliente();
    dgdGrid.DataSource =
cldCliente.Listar(txtDescricao.Text).Tables[0];
 }
```

Veja como o código fica muito simples: a classe clnCliente através do método Listar retorna um DataSet com o conteúdo requisitado e o controle DataGridView carrega automaticamente as informações.

As chamadas dos botões Novo e Alterar necessitam da criação de duas propriedades no formulário ***modelo de cadastro*** para a conexão entre os dois formulários. No *Solution Explorer* clique com o botão direito do mouse no formulário **frmModeloCadastro** e adicione as duas propriedades conforme abaixo:

```
clnFuncoesGerais.Operacao _Operacao;

        public clnFuncoesGerais.Operacao Operacao
        {
            get
            {
                return _Operacao;
            }
            set
            {
                _Operacao = value;
            }
        }

        int _Codigo;

        public int Codigo
        {
            get
            {
                return _Codigo;
            }
```



```
            set
            {
                _Codigo = value;
            }
        }
```

No modo design do formulário de cadastro **frmClienteConsulta**, dê um duplo clique sobre o botão **Novo** e adicione o seguinte código:

```
frmClienteCadastro frmCliente = new frmClienteCadastro();
frmCliente.Operacao = clnFuncoesGerais.Operacao.Inclusao;
frmCliente.ShowDialog();
CarregaGrid();
```

É feita a chamada ao formulário de cadastro passando como parâmetro a operação de inclusão.

Novamente no modo design do formulário, dê um duplo clique sobre o botão **Alterar** e adicione o seguinte código:

```
  frmClienteCadastro frmCliente = new frmClienteCadastro();
  frmCliente.Operacao = clnFuncoesGerais.Operacao.Alteracao;
  frmCliente.Codigo = (int)dgdGrid.CurrentRow.Cells[0].Value;
  frmCliente.ShowDialog();
  CarregaGrid();
```

A única diferença em relação ao botão Novo é a passagem do parâmetro Código para ser utilizado na alteração do produto como chave.

Novamente no modo design do formulário, dê um duplo clique sobre o botão Excluir e adicione o seguinte código:

```
if ((MessageBox.Show("Deseja excluir o registro?", this.Text,
MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.No))
    {
        return;
    }
  clnCliente cldCliente = new clnCliente();

cldCliente.Excluir((int)(dgdGrid.CurrentRow.Cells[0].Value));

   MessageBox.Show("Registro excluído com sucesso",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);
   CarregaGrid();
```

O código já faz a chamada diretamente à classe de cliente o método excluir e a utilização de mensagens de confirmação e de pergunta sobre a exclusão ou não do registro.

Para finalizar o formulário, dê um duplo clique no botão Sair e adicione o código abaixo:

```
this.Close();
```

O código simplesmente fecha o formulário.

Formulário de Cadastro

O formulário de cadastro efetua duas tarefas: inclusão e alteração. Essas tarefas dependem principalmente da validação dos dados, que será a primeira etapa do desenvolvimento.

Adicione as referências que seguem no código do formulário (*view code ou F7*):

```
using DllControleDeVendas.Sistema.Globais;
using DllControleDeVendas.Sistema.Negocio;
```

No formulário padrão foi adicionado um controle chamado ErrorProvider, cujo objetivo principal é facilitar a validação de dados em formulários. Utilizaremos o controle para validar o formulário.

No modo design do formulário, dê um duplo clique no botão ***Gravar*** e adicione o seguinte código:

```
    if (txtNome.Text.Equals(string.Empty))
            {
                errErro.SetError(txtNome, "Digite um nome");
                return;
            }
            else
            {
                errErro.SetError(txtNome, "");
            }
            if (txtCpfCnpj.Text.Equals(string.Empty))
            {
                errErro.SetError(txtCpfCnpj, "Digite um
CPF/CNPJ");
                return;
            }
            else
```



```
            {
                errErro.SetError(txtCpfCnpj, "");
            }
            if (txtLogradouro.Text.Equals(string.Empty))
            {
                errErro.SetError(txtLogradouro, "Digite um
logradouro");
                return;
            }
            else
            {
                errErro.SetError(txtLogradouro, "");
            }
            if (txtBairro.Text.Equals(string.Empty))
            {
                errErro.SetError(txtBairro, "Digite um
bairro");
                return;
            }
            else
            {
                errErro.SetError(txtBairro, "");
            }
            if (txtCidade.Text.Equals(string.Empty))
            {
                errErro.SetError(txtCidade, "Digite uma
cidade");
                return;
            }
            else
            {
                errErro.SetError(txtCidade, "");
            }
            if (cboUf.Text.Equals(string.Empty))
            {
                errErro.SetError(cboUf, "Selecione uma UF");
                return;
            }
            else
            {
                errErro.SetError(cboUf, "");
            }
            if (txtCep.Text.Equals(string.Empty))
            {
                errErro.SetError(txtCep, "Digite um CEP");
                return;
```

```
            }
            else
            {
                errErro.SetError(txtCep, "");
            }
```

O controle ErrorProvider possui o método SetError, que já exibe um alerta com a mensagem especificada no parâmetro, facilitando assim a validação de erro. Após a validação, basta passar todos os parâmetros dos campos e a chamada do método gravar/alterar.

```
clnCliente cldCliente = new clnCliente();
            cldCliente.cli_Bairro = txtBairro.Text;
            cldCliente.cli_CEP = txtCep.Text;
            cldCliente.cli_Cidade = txtCidade.Text;
            cldCliente.cli_CNPJCPF = txtCpfCnpj.Text;
            cldCliente.cli_DtCadastro = DateTime.Now;
            cldCliente.cli_Email = txtEmail.Text;
            cldCliente.cli_Fones = txtTelefone.Text;
            cldCliente.cli_Logradouro = txtLogradouro.Text;
            cldCliente.cli_NomeRazao = txtNome.Text;
            cldCliente.cli_Uf = cboUf.Text;
            if (Operacao ==
clnFuncoesGerais.Operacao.Inclusao)
            {
                cldCliente.Gravar();
            }
            else if (Operacao ==
clnFuncoesGerais.Operacao.Alteracao)
            {
                cldCliente.cli_Id = Codigo;
                cldCliente.Alterar();
            }
            MessageBox.Show("Registro gravado com sucesso",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Close();
        }
```

A passagem de parâmetros é muito simples e no final a chamada do método Gravar. Com a confirmação da gravação, um alerta é exibido e o formulário é fechado.

A última tarefa a ser efetuada é o preenchimento do formulário no momento da chamada da alteração dos dados. Esse preenchimento será realizado no momento da abertura do formulário do cadastro; através de uma consulta os dados serão car-

regados. Abra novamente o formulário no modo design e dê um duplo clique sobre o fundo do formulário; é aberto o evento *FormLoad*, evento que é executado no momento da abertura do formulário. Adicione o código que segue no evento:

```
if (Operacao == clnFuncoesGerais.Operacao.Alteracao)
 {
  clnCliente clncliente = new clnCliente();
 System.Data.OleDb.OleDbDataReader drDados;
 drDados = clncliente.ListarCliente(Codigo);
 if (drDados.HasRows)
 {
  drDados.Read();
  txtNome.Text = drDados["cli_NomeRazao"].ToString();
  txtCpfCnpj.Text = drDados["cli_CNPJCPF"].ToString();
  txtLogradouro.Text = drDados["cli_Logradouro"].ToString();
  txtBairro.Text = drDados["cli_Bairro"].ToString();
  txtCidade.Text = drDados["cli_Cidade"].ToString();
  cboUf.Text = drDados["cli_Uf"].ToString();
  txtCep.Text = drDados["cli_CEP"].ToString();
  txtEmail.Text = drDados["cli_Email"].ToString();
  txtTelefone.Text = drDados["cli_Fones"].ToString();
 }
}
```

É carregado um DataReader que preenche todos os campos do retorno da função ListarCliente.

Finalmente o primeiro formulário está pronto. Com isso, podemos testar se o cadastro está realmente funcionando. Antes de executar o formulário é necessário adicionar a chamada do formulário de cadastro no menu principal do sistema. Abra o formulário de menu ***frmMenu*** no modo design, abra o menu e dê um duplo clique sobre o item do menu de cliente. Adicione o código de chamada:

```
frmClienteConsulta frmCliente = new frmClienteConsulta();
frmCliente.ShowDialog();
```

O código é muito simples. É instanciado o formulário e chamado o método ShowDialog para a exibição do formulário. Execute a aplicação (F5) ou no menu Debug e Start.

Abra no menu o formulário de clientes e execute as funcionalidades implementadas.

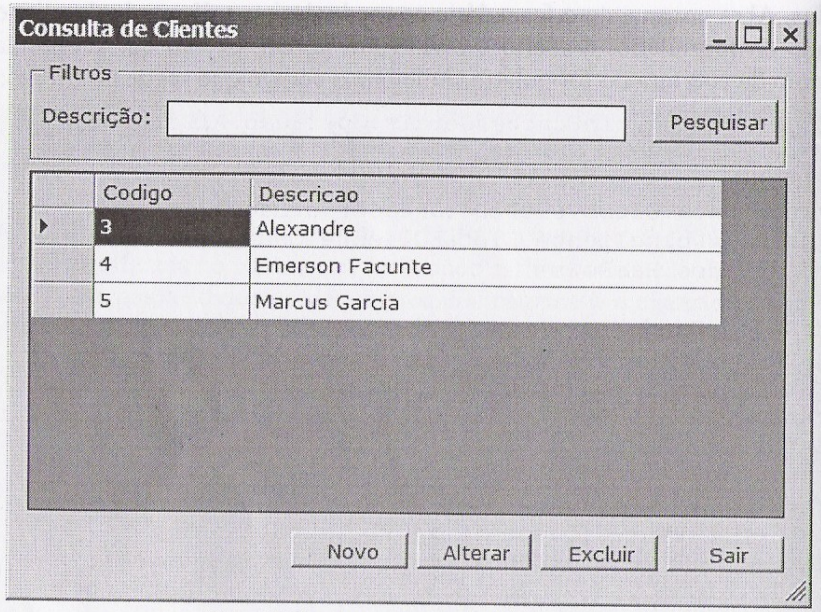


Figura 11.20 – Formulário de consulta de cliente.

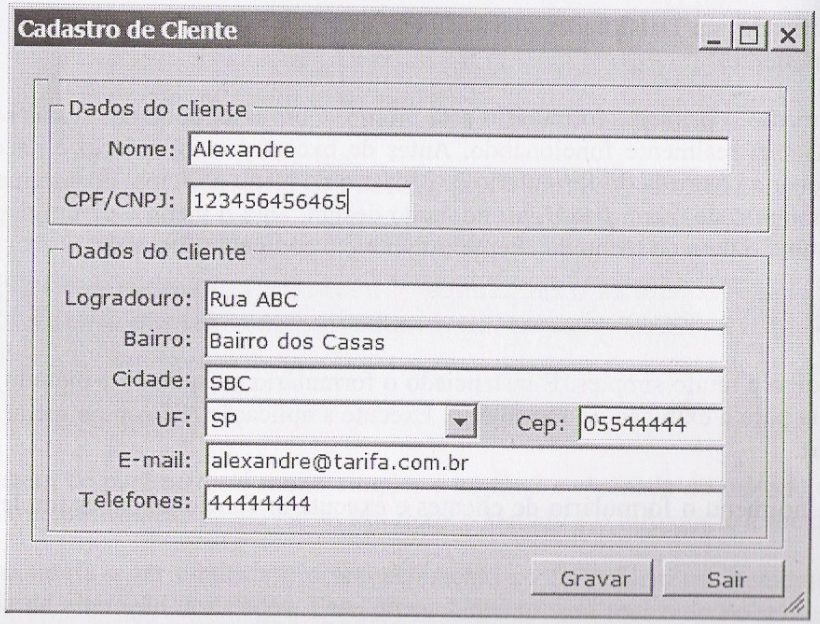


Figura 11.21 – Formulário de cadastro de cliente.

Cadastro de Categoria

O cadastro de categoria possui as funcionalidades idênticas ao de cliente, dois formulários, consulta e cadastro. O primeiro a ser implementado será o de consulta, que será responsável pela chamada do formulário de cadastro.

Formulário de Consulta (Categoria)

No *Solution Explorer* dê um duplo clique no formulário frmCategoriaConsulta e o modo design do formulário é aberto.

Adicione as referências que seguem no código do formulário (*view code ou F7*):

```
using DllControleDeVendas.Sistema.Globais;
using DllControleDeVendas.Sistema.Negocio;
```

Crie a primeira funcionalidade do formulário, a rotina CarregaGrid. Para ser exibido o código do formulário, clique com o botão direito do mouse sobre o formulário e *view code* ou pressione F7. Adicione o código conforme abaixo:

```
private void CarregaGrid()
{
  clnCategoria cldCategoria = new clnCategoria();
  dgdGrid.DataSource =
cldCategoria.Listar(txtDescricao.Text).Tables[0];
}
```

Veja como o código fica muito simples: o classe clnCategoria através do método Listar retorna um DataSet com o conteúdo requisitado e o controle DataGridView carrega automaticamente as informações.

No modo design do formulário de cadastro **frmCategoriaConsulta**, dê um duplo clique sobre o botão **Novo** e adicione o seguinte código:

```
frmCategoriaCadastro frmCategoria = new
frmCategoriaCadastro();

frmCategoria.Operacao = clnFuncoesGerais.Operacao.Inclusao;
frmCategoria.ShowDialog();

CarregaGrid();
```

É feita a chamada ao formulário de cadastro passando como parâmetro a operação de inclusão.

Novamente no modo design do formulário, dê um duplo clique sobre o botão **Alterar** e adicione o seguinte código:

```
frmCategoriaCadastro frmCategoria = new
frmCategoriaCadastro();
frmCategoria.Operacao = clnFuncoesGerais.Operacao.Alteracao;
frmCategoria.Codigo = (int)dgdGrid.CurrentRow.Cells[0].Value;
frmCategoria.ShowDialog();
CarregaGrid();
```

A única diferença em relação ao botão Novo é a passagem do parâmetro Código para ser utilizado na alteração do produto como chave.

Novamente no modo design do formulário, dê um duplo clique sobre o botão Excluir e adicione o seguinte código:

```
if ((MessageBox.Show("Deseja excluir o registro?", this.Text,
MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.No))
 {
  return;
  }
clnCategoria cldCategoria = new clnCategoria();

cldCategoria.Excluir((int)(dgdGrid.CurrentRow.Cells[0].Value)
);

MessageBox.Show("Registro excluído com sucesso", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Information);
 CarregaGrid();
```

O código já faz a chamada diretamente à classe de categoria o método excluir e a utilização de mensagens de confirmação e de pergunta sobre a exclusão ou não do registro.

Para finalizar o formulário, dê um duplo clique no botão Sair e adicione o código abaixo:

```
this.Close();
```

O código simplesmente fecha o formulário.

Formulário de Cadastro (Categoria)

O formulário de cadastro efetua duas tarefas: inclusão e alteração. Essas tarefas dependem principalmente da validação dos dados, que será a primeira etapa do desenvolvimento.

Adicione as referências que seguem no código do formulário (*view code ou F7)*:

```
using DllControleDeVendas.Sistema.Globais;
using DllControleDeVendas.Sistema.Negocio;
```

Conforme o cadastro de cliente, a validação é efetuada utilizando o controle ErrorProvider do formulário modelo.

No modo design do formulário, dê um duplo clique no botão **Gravar** e adicione o seguinte código:

```
if (txtNome.Text.Equals(string.Empty))
{
  errErro.SetError(txtNome, "Digite um NOME");
  return;
}
else
{
  errErro.SetError(txtNome, "");
}
```

O controle ErrorProvider possui o método SetError que já exibe um alerta com a mensagem especificada no parâmetro, facilitando assim a validação de erro. Após a validação, basta passar todos os parâmetros dos campos e a chamada do método gravar/alterar.

```
clnCategoria cldCategoria = new clnCategoria();
cldCategoria.cat_Descricao = txtNome.Text;

 if (Operacao == clnFuncoesGerais.Operacao.Inclusao)
 {
   cldCategoria.Gravar();
 }
 else
 if (Operacao == clnFuncoesGerais.Operacao.Alteracao)
 {
   cldCategoria.cat_ID = Codigo;
   cldCategoria.Alterar();
 }
```

```
 MessageBox.Show("Registro gravado com sucesso", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Close();
```

A passagem de parâmetros é muito simples e no final a chamada do método Gravar/Alterar. Com a confirmação da gravação, um alerta é exibido e o formulário é fechado.

A última tarefa a ser efetuada é o preenchimento do formulário no momento da chamada da alteração dos dados. Esse preenchimento será realizado no momento da abertura do formulário do cadastro e através de uma consulta os dados serão carregados. Abra novamente o formulário no modo design e dê um duplo clique sobre o fundo do formulário; é aberto o evento **FormLoad**, evento que é executado no momento da abertura do formulário. Adicione o código no evento:

```
if (Operacao == clnFuncoesGerais.Operacao.Alteracao)
 {
   clnCategoria clnCategoria = new clnCategoria();
   System.Data.OleDb.OleDbDataReader drDados;
   drDados = clnCategoria.ListarCategoria(Codigo);
  if (drDados.HasRows)
  {
    drDados.Read();
    txtNome.Text = drDados["cat_Descricao"].ToString();
  }
}
```

É carregado um DataReader que preenche todos os campos do retorno da função ListarCategoria.

Finalmente o formulário está pronto. Com isso, podemos testar se o cadastro está realmente funcionando. Antes de executar o formulário é necessário adicionar a chamada do formulário de cadastro no menu principal do sistema. Abra o formulário de menu **frmMenu** no modo design, abra o menu e dê um duplo sobre o item do menu de categoria. Adicione o código de chamada:

```
frmCategoriaConsulta frmCategoria = new
frmCategoriaConsulta();
            frmCategoria.ShowDialog();
```

O código é muito simples, basicamente é instanciado o formulário e chamado o método ShowDialog para a exibição do formulário. Execute a aplicação (F5) ou no menu Debug e Start.

Abra no menu o formulário de categoria e execute as funcionalidades implementadas.

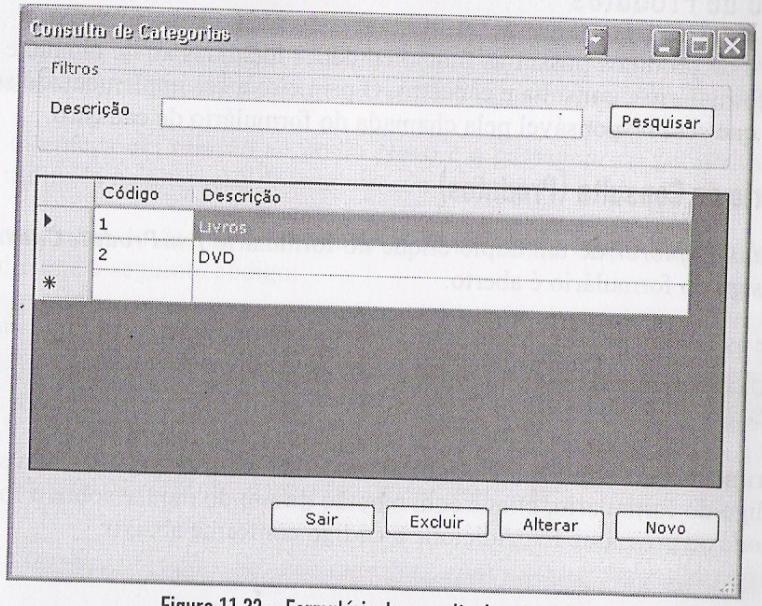


Figura 11.22 – Formulário de consulta de categoria.

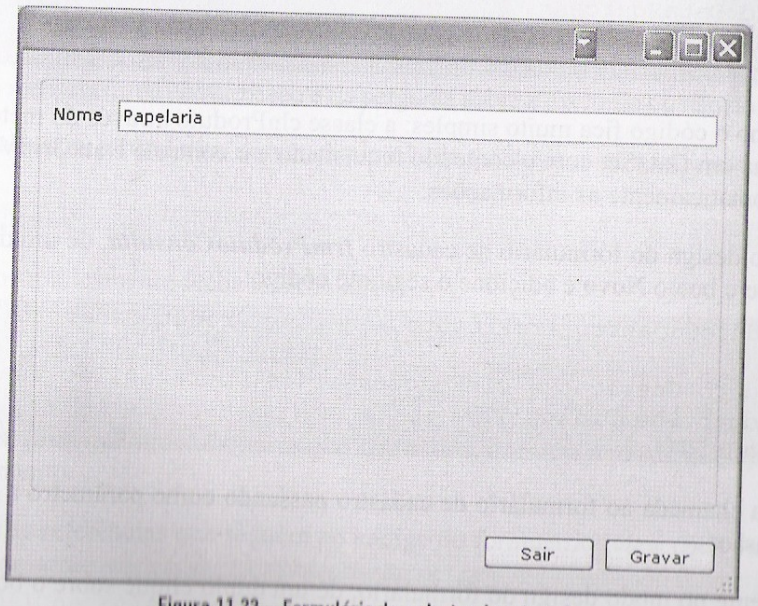


Figura 11.23 – Formulário de cadastro de categoria.

Cadastro de Produtos

O cadastro de produtos possui as funcionalidades idênticas ao de cliente e categoria, dois formulários, consulta e cadastro. O primeiro a ser implementado será o de consulta, que será responsável pela chamada do formulário de cadastro.

Formulário de Consulta (Produtos)

No *Solution Explorer* dê um duplo clique no formulário ***frmProdutoConsulta***, e o modo design do formulário é aberto.

Adicione as referências que seguem no código do formulário (*view code ou F7*):

```
using DllControleDeVendas.Sistema.Globais;
using DllControleDeVendas.Sistema.Negocio;
```

Crie a primeira funcionalidade do formulário, a rotina CarregaGrid. Para ser exibido o código do formulário, clique com o botão direito do mouse sobre o formulário e *view code* ou pressione F7. Adicione o código conforme abaixo:

```
private void CarregaGrid()
 {
  clnProduto cldProduto = new clnProduto();
 dgdGrid.DataSource =
cldProduto.Listar(txtDescricao.Text).Tables[0];
 }
```

Veja como o código fica muito simples: a classe clnProduto através do método Listar retorna um DataSet com o conteúdo requisitado e o controle DataGridView carrega automaticamente as informações.

No modo design do formulário de cadastro ***frmProdutoConsulta***, dê um duplo clique sobre o botão **Novo** e adicione o seguinte código:

```
frmProdutoCadastro frmProduto = new frmProdutoCadastro();

frmProduto.Operacao = clnFuncoesGerais.Operacao.Inclusao;
frmProduto.ShowDialog();
CarregaGrid();
```

É feita a chamada ao formulário de cadastro passando como parâmetro a operação de inclusão.

Novamente no modo design do formulário, dê um duplo clique sobre o botão **Alterar** e adicione o seguinte código:


```
frmProdutoCadastro frmProduto = new frmProdutoCadastro();
frmProduto.Operacao = clnFuncoesGerais.Operacao.Alteracao;
frmProduto.Codigo = (int)dgdGrid.CurrentRow.Cells[0].Value;
frmProduto.ShowDialog();
CarregaGrid();
```

A única diferença em relação ao botão Novo é a passagem do parâmetro Código para ser utilizado na alteração do produto como chave.

Novamente no modo design do formulário, dê um duplo clique sobre o botão Excluir e adicione o seguinte código:

```
if ((MessageBox.Show("Deseja excluir o registro?", this.Text,
MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.No))
 {
   return;
 }
clnProduto cldProduto = new clnProduto();

cldProduto.Excluir((int)(dgdGrid.CurrentRow.Cells[0].Value));

MessageBox.Show("Registro excluído com sucesso", this.Text,
MessageBoxButtons.OK, MessageBoxIcon.Information);
CarregaGrid();
```

O código já faz a chamada diretamente à classe de produto o método excluir e a utilização de mensagens de confirmação e de pergunta sobre a exclusão ou não do registro.

Para finalizar o formulário, dê um duplo clique no botão sair e adicione o código abaixo:

```
this.Close();
```

O código simplesmente fecha o formulário.

Formulário de Cadastro (Produtos)

O formulário de cadastro efetua duas tarefas, inclusão e alteração. Essas tarefas dependem principalmente da validação dos dados, que será a primeira etapa do desenvolvimento.

Adicione as referências que seguem no código do formulário (*view code ou F7*):

```
using DllControleDeVendas.Sistema.Globais;
using DllControleDeVendas.Sistema.Negocio;
```

Conforme o cadastro de produto, a validação é efetuada utilizando o controle ErrorProvider do formulário modelo.

No modo design do formulário, dê um duplo clique no botão **Gravar** e adicione o seguinte código:

```
if (cboCategoria.Text.Equals(string.Empty))
            {
                errErro.SetError(cboCategoria, "Informe uma
Categoria");
                return;
            }
            else
            {
                errErro.SetError(cboCategoria, "");
            }
            if (txtDescricao.Text.Equals(string.Empty))
            {
                errErro.SetError(txtDescricao, "Digite uma
Descrição");
                return;
            }
            else
            {
                errErro.SetError(txtDescricao, "");
            }
            if (nudQuantidade.Text.Equals(string.Empty))
            {
                errErro.SetError(nudQuantidade, "Digite uma
Quantidade");
                return;
            }
            else
            {
                errErro.SetError(nudQuantidade, "");
            }
            if (txtValor.Text.Equals(string.Empty))
            {
                errErro.SetError(txtValor, "Digite um Valor");
                return;
            }
            else
            {
                errErro.SetError(txtValor, "");
            }
```


O controle ErrorProvider possui o método SetError que já exibe um alerta com a mensagem especificada no parâmetro, facilitando assim a validação de erro. Após a validação, basta passar todos os parâmetros dos campos e a chamada do método Gravar/Alterar.

Ainda no código do botão gravar, digite o código que segue.

```
clnCategoria clnCategoria = new clnCategoria();
            clnCategoria.cat_ID =
int.Parse(cboCategoria.SelectedValue.ToString());

            clnProduto clnProduto = new clnProduto();
            clnProduto.clnCategoria = clnCategoria;

            clnProduto.pro_Descricao = txtDescricao.Text;
            clnProduto.pro_QtdeEstoque =
Convert.ToInt16(nudQuantidade.Text);
            clnProduto.pro_Valor =
Convert.ToDecimal(txtValor.Text);
            clnProduto.pro_Ativo = optSim.Checked;


            if (Operacao ==
clnFuncoesGerais.Operacao.Inclusao)
            {
                clnProduto.Gravar();
            }
            else if (Operacao ==
clnFuncoesGerais.Operacao.Alteracao)
            {
                clnProduto.pro_id = Codigo;
                clnProduto.Alterar();
            }
            MessageBox.Show("Registro gravado com sucesso",
this.Text, MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Close();
```

A passagem de parâmetros é muito simples e no final temos a chamada do método Gravar/Alterar. Com a confirmação da gravação, um alerta é exibido e o formulário é fechado. A última tarefa a ser efetuada é o preenchimento do formulário no momento da chamada da alteração dos dados. Esse preenchimento será realizado no momento da abertura do formulário do cadastro; através de uma consulta os dados serão carregados. Abra novamente o formulário no modo design e dê um duplo clique sobre o fundo do formulário. É aberto o evento **FormLoad**, evento que é executado no momento da abertura do formulário. Adicione o código no evento:

```
clnCategoria clnCategoria = new clnCategoria();
DataSet dsCategoria = new DataSet();
dsCategoria = clnCategoria.Listar("");
cboCategoria.DataSource = dsCategoria.Tables[0];
cboCategoria.DisplayMember = "Descricao";
cboCategoria.ValueMember = "Codigo";
if (Operacao == clnFuncoesGerais.Operacao.Alteracao)
{
  clnProduto clnProduto = new clnProduto();
  System.Data.OleDb.OleDbDataReader drDados;
  drDados = clnProduto.ListarProduto(Codigo);
  if (drDados.HasRows)
  {
   drDados.Read();
   cboCategoria.SelectedValue =
drdados["cat_Codigo"].ToString();
   txtDescricao.Text = drDados["pro_Descricao"].ToString();
   nudQuantidade.Text =
drDados["pro_QtdeEstoque"].ToString();
   txtValor.Text = drDados["pro_Valor"].ToString();
   if (drDados["pro_Ativo"].ToString() == "true")
   {
      optSim.Checked = true;
   }
   else
   {
      optNao.Checked = true;
   }
  }
}
```

É carregado um DataReader que preenche todos os campos do retorno da função ListarProduto e é carregado o ComboBox com todas as categorias cadastradas.

Finalmente o formulário está pronto, com isso podemos testar se está realmente funcionando o cadastro. Antes de executar o formulário é necessário adicionar a chamada do formulário de cadastro no menu principal do sistema. Abra o formulário de menu frmMenu no modo design, abra o menu e dê um duplo sobre o item do menu de produto. Adicione o código de chamada:

```
frmProdutoConsulta frmProduto = new frmProdutoConsulta();
frmProduto.ShowDialog();
```


O código é muito simples, basicamente é instanciado o formulário e chamado o método ShowDialog para a exibição do formulário. Execute a aplicação (F5) ou no menu Debug e Start.

Abra no menu o formulário de produtos e execute as funcionalidades implementadas.

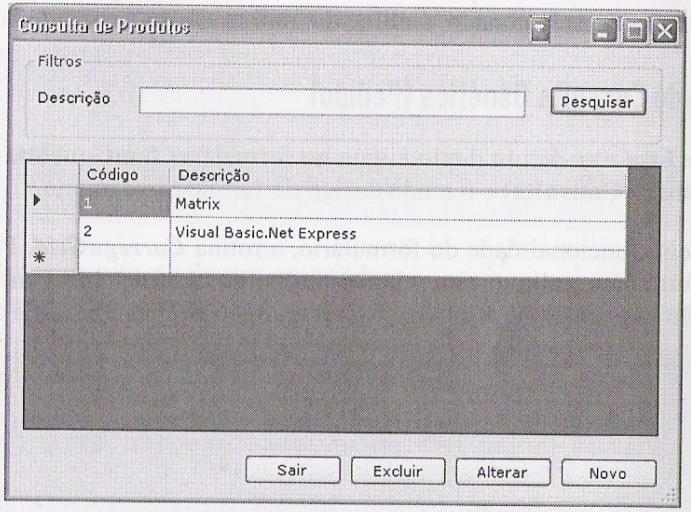


Figura 11.24 – Formulário de consulta de produto.

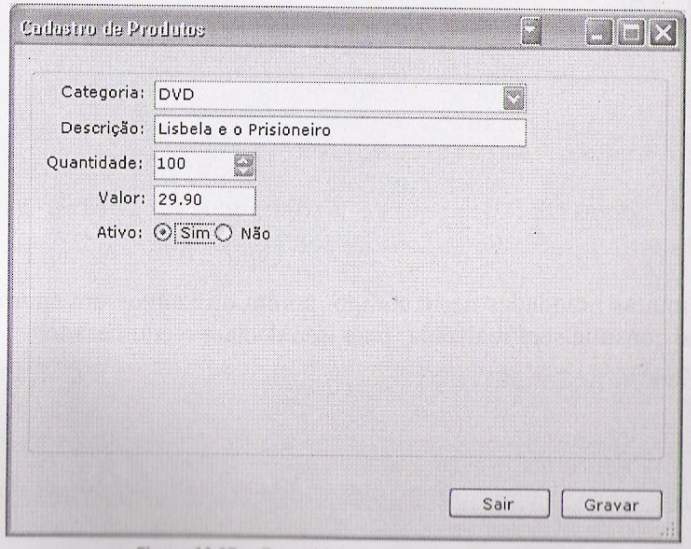


Figura 11.25 – Formulário de cadastro de produto.

Cadastro de Pedido

O cadastro de pedido possui funcionalidades diferentes dos formulários anteriores. Este formulário é o mais complexo do sistema, envolvendo diversas funcionalidades. Temos um formulário de pedido e um outro de consulta genérica de cliente ou produto que, quando selecionado, é utilizado no formulário de pedido.

Formulário de Consulta Genérica (Pedido)

No *Solution Explorer* dê um duplo clique no formulário **frmConsulta (Formulários > Sistema > Consulta)** e o modo design do formulário é aberto.

Crie a primeira funcionalidade do formulário, a rotina ***CarregaGrid.*** Para exibir o código do formulário, clique com o botão direito do mouse sobre o mesmo e selecione a opção *view code* ou pressione F7. Adicione o código conforme abaixo:

```
private void CarregaGrid()
        {
            DataSet dsConsulta = new DataSet();
            if (_ConsultaTipo == TipoConsulta.Cliente)
            {
                clnCliente clnCliente = new clnCliente();
                dsConsulta =
clnCliente.Listar(txtDescricao.Text);
            }
            else if (_ConsultaTipo == TipoConsulta.Produto)
            {
                clnProduto clnProduto = new clnProduto();
                dsConsulta =
clnProduto.Listar(txtDescricao.Text);
            }
            dgdGrid.DataSource = dsConsulta.Tables[0];
        }
```

Não temos muitas novidades neste código, porém utilizamos um enumerador para detectar qual consulta será realizada; para isso declare o enumerador:

```
public enum TipoConsulta
        {
            Cliente,
            Produto
        }
```


Vamos codificar o botão ***Pesquisar.***

```
    CarregaGrid();
```

O código atualiza o Grid de acordo com a nova pesquisa.

Vamos codificar o botão ***Selecionar***:

```
 _CodigoRetorno =
        Convert.ToInt16(dgdGrid.CurrentRow.Cells[0].Value);
 _DescricaoRetorno =
        (dgdGrid.CurrentRow.Cells[0].Value.ToString());

 Close();
```

As propriedades de retorno devem ser criadas:

```
private TipoConsulta _ConsultaTipo;

public TipoConsulta ConsultaTipo {
 get {
   return _ConsultaTipo;
 }
 set {
   _ConsultaTipo = value;
 }
}

private int _CodigoRetorno;

public int CodigoRetorno {
 get {
   return _CodigoRetorno;
 }
 set {
   _CodigoRetorno = value;
 }
}

private string _DescricaoRetorno;

public string DescricaoRetorno {
 get {
   return _DescricaoRetorno;
 }
 set {
```

```
    _DescricaoRetorno = value;
  }
}
```

Para finalizar o formulário, dê um duplo clique no botão sair e adicione o código abaixo:

```
this.Close();
```

O código simplesmente fecha o formulário.

Formulário de Pedido

A seleção de cliente e produto é feita pela tela genérica de consulta, bastando realizar a chamada ao formulário. Dê um duplo clique sobre o botão buscar cliente e entre com o seguinte código:

```
frmConsulta frmConsulta = new frmConsulta();
frmConsulta.Text = "Consulta de Cliente";
frmConsulta.ConsultaTipo = frmConsulta.TipoConsulta.Cliente;
frmConsulta.ShowDialog();


CodigoCliente = frmConsulta.CodigoRetorno;
txtCliente.Text = frmConsulta.DescricaoRetorno;
```

A consulta de produto é bem semelhante, só mudando o enumerador de tipo de consulta:

```
frmConsulta frmConsulta = new frmConsulta();
frmConsulta.Text = "Consulta de Produgto";
frmConsulta.ConsultaTipo = frmConsulta.TipoConsulta.Produto;
frmConsulta.ShowDialog();


CodigoProduto = frmConsulta.CodigoRetorno;
txtProduto.Text = frmConsulta.DescricaoRetorno;
```

O formulário de pedido efetua duas tarefas: adicionar pedido e adicionar item de pedido. Essas tarefas dependem principalmente da validação dos dados, que será a primeira etapa do desenvolvimento.

A validação é efetuada utilizando o controle ErrorProvider do formulário modelo.

No modo design do formulário, dê um duplo clique no botão Adicionar no grupo de Pedido e adicione o seguinte código:

```
if (txtCliente.Text == String.Empty)
{
    errErro.SetError(txtCliente, "Selecione um cliente");
}
else
{
    errErro.SetError(txtCliente, "");
}
clnCliente clnCliente = new clnCliente();
clnCliente.cli_Id = CodigoCliente;

clnPedido clnPedido = new clnPedido();
clnPedido.clnCliente = clnCliente;
clnPedido.ped_DtEncomenta = dtpData.Value;
clnPedido.ped_Valor = 0;
clnPedido.ped_Status = 'A';
CodigoPedido = clnPedido.Gravar();

MessageBox.Show("Pedido adicionado com sucesso");
grbItem.Enabled = true;
grbPedido.Enabled = true;
```

O controle ErrorProvider possui o método SetError, que já exibe um alerta com a mensagem especificada no parâmetro, facilitando assim a validação de erro. Após a validação, basta passar todos os parâmetros dos campos e a chamada do método gravar.

Neste ponto vamos codificar o botão ***Adicionar*** do grupo ***Item.***

```
// verifica se controle txtProduto está vazio
if (txtProduto.Text == String.Empty)
{
   errErro.SetError(txtProduto, "Selecione um produto");
}
else
{
   errErro.SetError(txtProduto, "");
}

// verifica se controle txtQuantidade está vazio
if (txtQuantidade.Text == String.Empty)
{
   errErro.SetError(txtQuantidade, "Informe a quantidade");
```

```
}
else
{
    errErro.SetError(txtQuantidade, "");
}

if (txtValor.Text == String.Empty)
{
    errErro.SetError(txtValor, "Informe o valor!");
}
else
{
    errErro.SetError(txtValor, "");
}

// obtem código do pedido
clnPedido clnPedido = new clnPedido();
clnPedido.ped_id = CodigoPedido;

// obtem código do produto
clnProduto clnProduto = new clnProduto();
clnProduto.pro_id = CodigoProduto;

// relaciona item ao pedido
clnItem clnItem = new clnItem();
clnItem.clnPedido = clnPedido;
clnItem.clnProduto = clnProduto;
clnItem.ite_Qtde = Convert.ToInt16(txtQuantidade.Text);
clnItem.ite_Valor = Convert.ToDecimal(txtValor.Text);

// grava informações no banco de dados
clnItem.Gravar();

MessageBox.Show("Item adicionado com sucesso!");

// limpa controles
txtQuantidade.Text = "";
txtValor.Text = "";
txtProduto.Text = "";
CodigoProduto = 0;

// Atualiza Grid
CarregaGridItem();
```


É possível também excluir um item já adicionado no botão Remover. Dê um duplo clique sobre o botão Remover e adicione o código:

```
clnItem clnItem = new clnItem();

clnItem.Excluir(Convert.ToInt16(dgvItem.CurrentRow.Cells[0].V
alue));

MessageBox.Show("Item excluído com sucesso!");

CarregaGridItem();
```

Finalmente o formulário está pronto. Com isso podemos testar se a tela de pedido está realmente funcionando. Antes de executar o formulário é necessário adicionar a chamada do formulário de pedido no menu principal do sistema. Abra o formulário de menu frmMenu no modo design, abra o menu e dê um duplo clique sobre o item do menu de venda. Adicione o código de chamada:

```
frmPedido frmPedido = new frmPedido();
frmPedido.ShowDialog();
```

O código é muito simples, basicamente é instanciado o formulário e chamado o método ShowDialog para a exibição do formulário. Execute a aplicação (F5) ou no menu Debug e Start.

Abra no menu o formulário de Vendas e execute as funcionalidades implementadas.

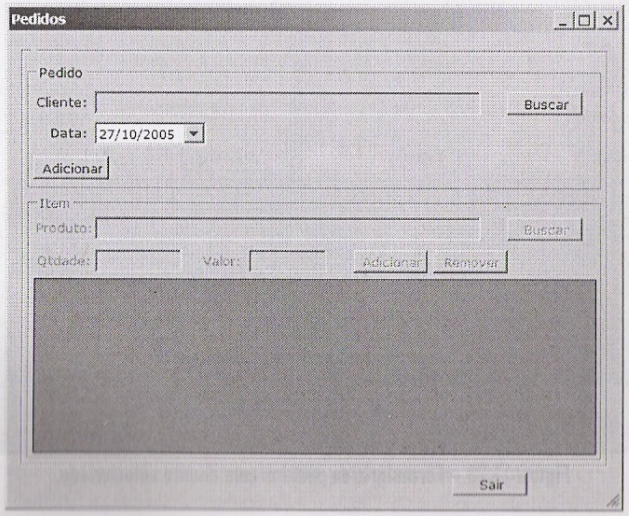


Figura 11.26 – Formulário de pedidos.

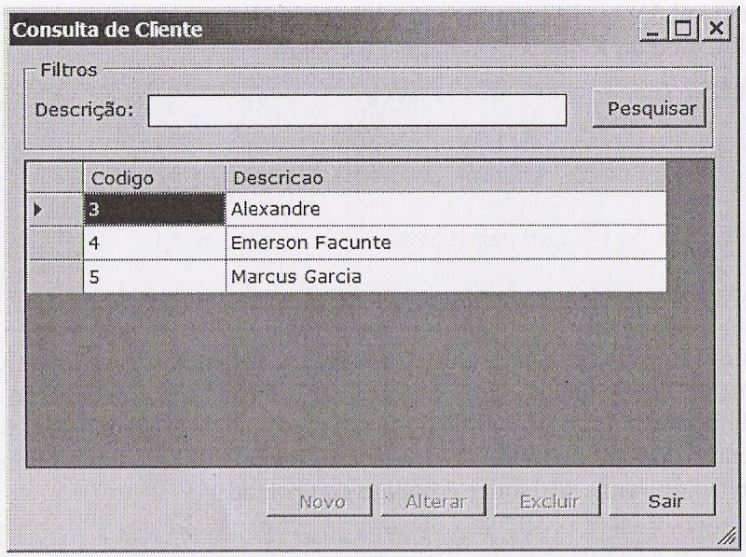


Figura 11.27 – Formulário de consulta de cliente.

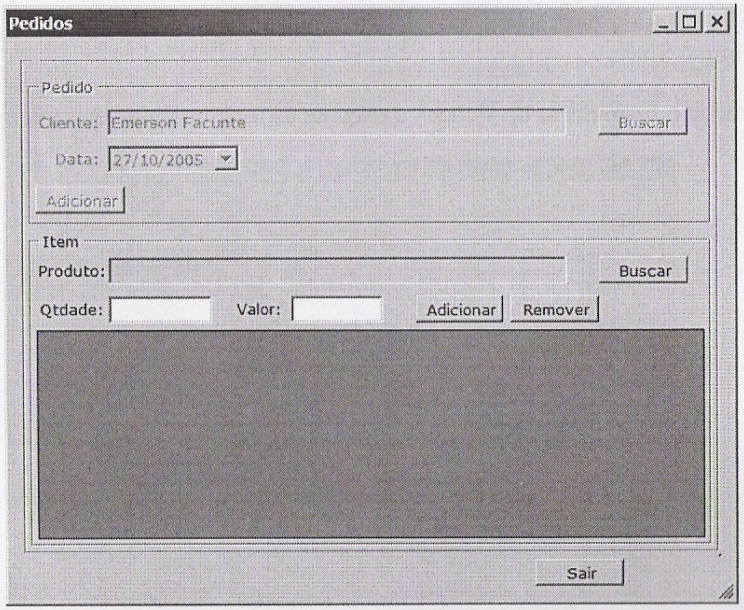


Figura 11.28 – Formulário de pedidos com cliente selecionado.

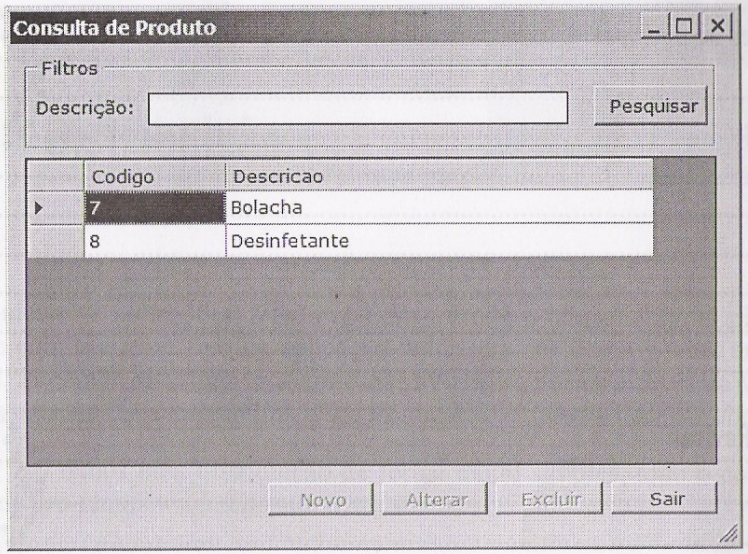


Figura 11.29 – Formulário de consulta de produto.

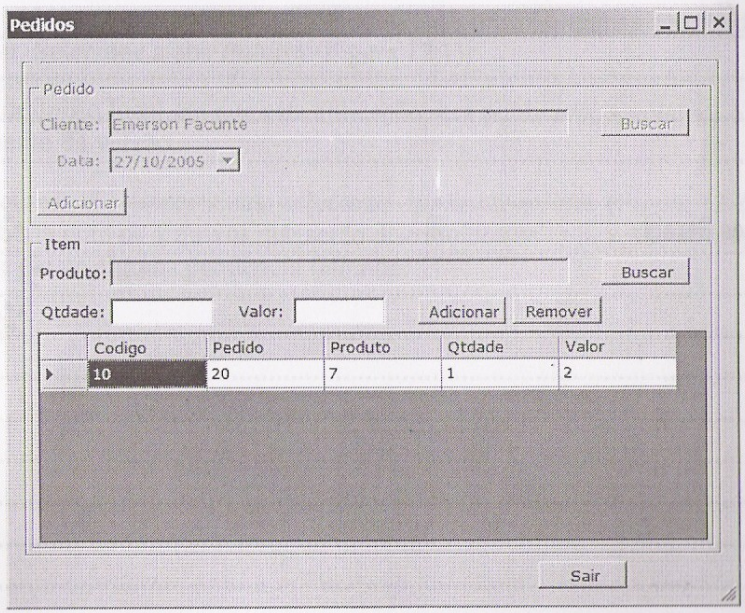


Figura 11.30 – Formulário de pedido com item adicionado.

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

CAPÍTULO 3

Distribuição

A distribuição de aplicações Windows Forms desde o lançamento da plataforma .NET fez com que esse tipo de aplicação voltasse a ser desenvolvida em muitas empresas. Podemos distribuir uma aplicação Windows Forms através da Web. Isso mesmo, o usuário clica em um link e um formulário Windows Forms é aberto.

No .NET 2.0 temos essa distribuição de forma muito simples, todo o controle de versão e de pré-requisitos é feito automaticamente.

Algumas configurações

Antes de publicar uma aplicação, podemos customizar algumas propriedades em *My Project*. Selecione a aba *Publish* (Figura 12.1).

Podemos definir o local em que esta aplicação será publicada, a versão e o controle de incremento da versão.

Outros recursos são importantes: selecione *Application Files* (Figura 12.2), e uma janela é aberta com os arquivos que serão distribuídos na aplicação, assim podemos conferir se existe alguma pendência faltando.

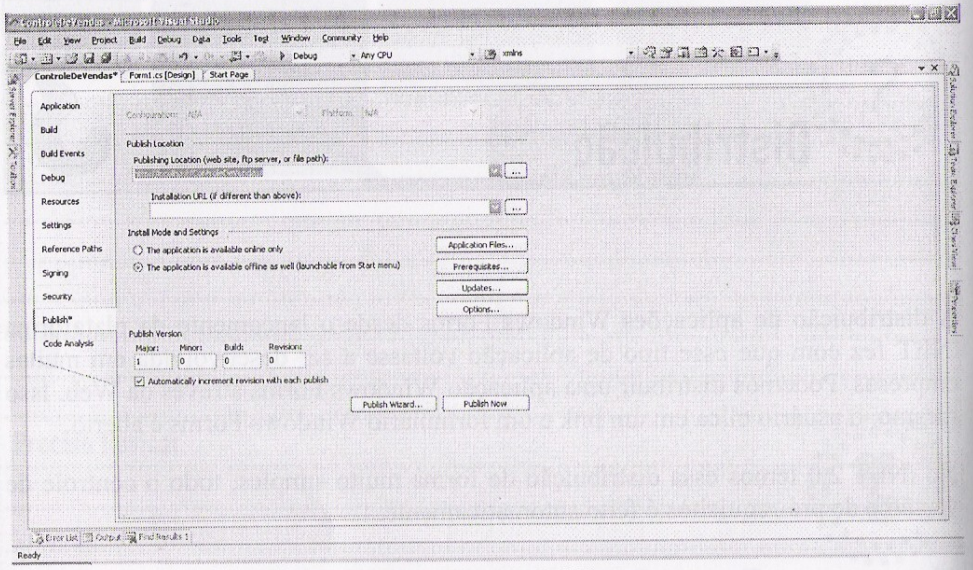


Figura 12.1 – Aba *Publish*.

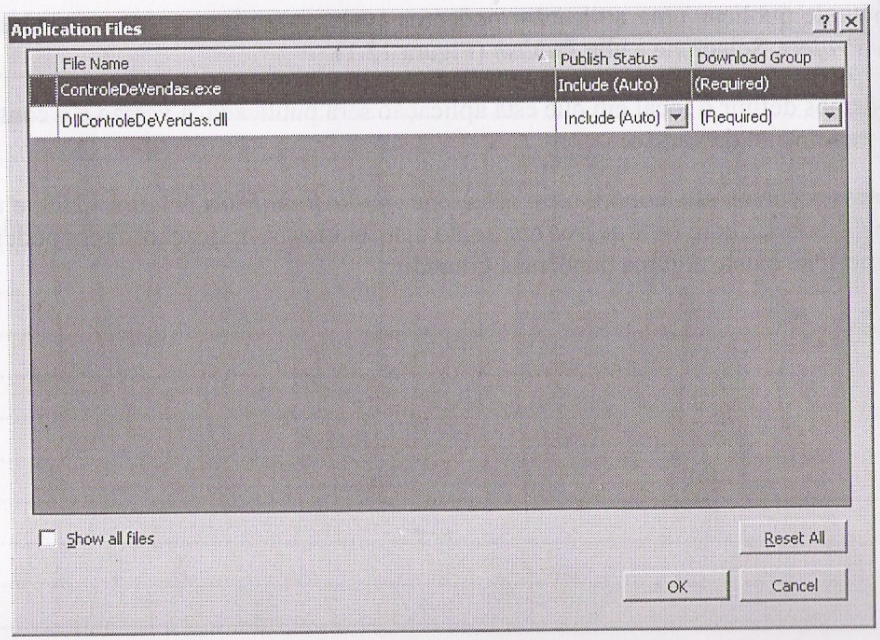


Figura 12.2 – Application Files

Clique em OK, e selecione *Prerequisites* (Figura 12.3).

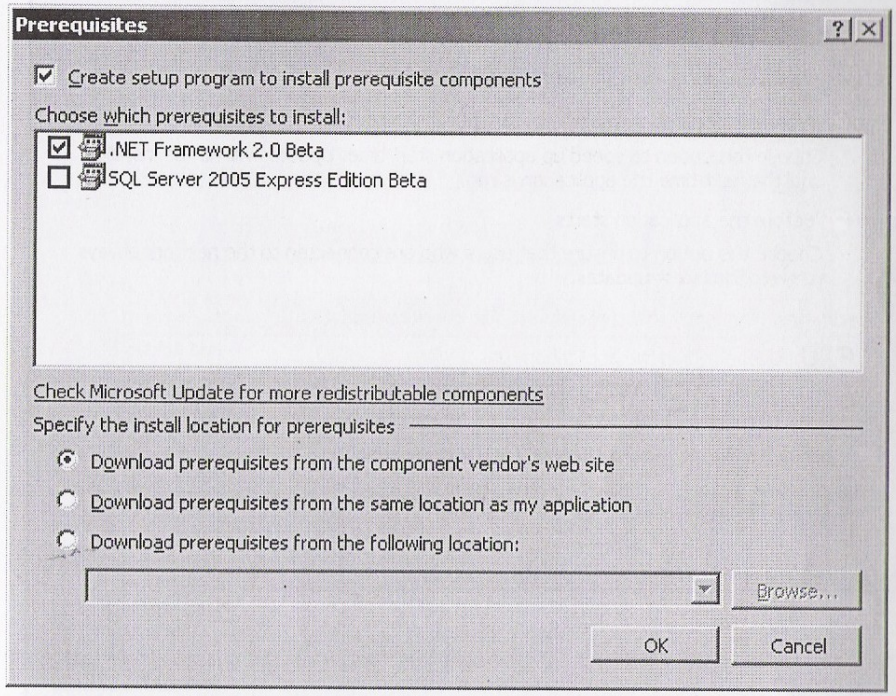


Figura 12.3 – Prerequisites

Nesta janela configuramos quais são os pré-requisitos de instalação na máquina do cliente e podemos definir se esses softwares dependentes serão baixados via download e instalados antes da aplicação, caso necessário.

Clique em OK e selecione *Updates* (Figura 12.4).

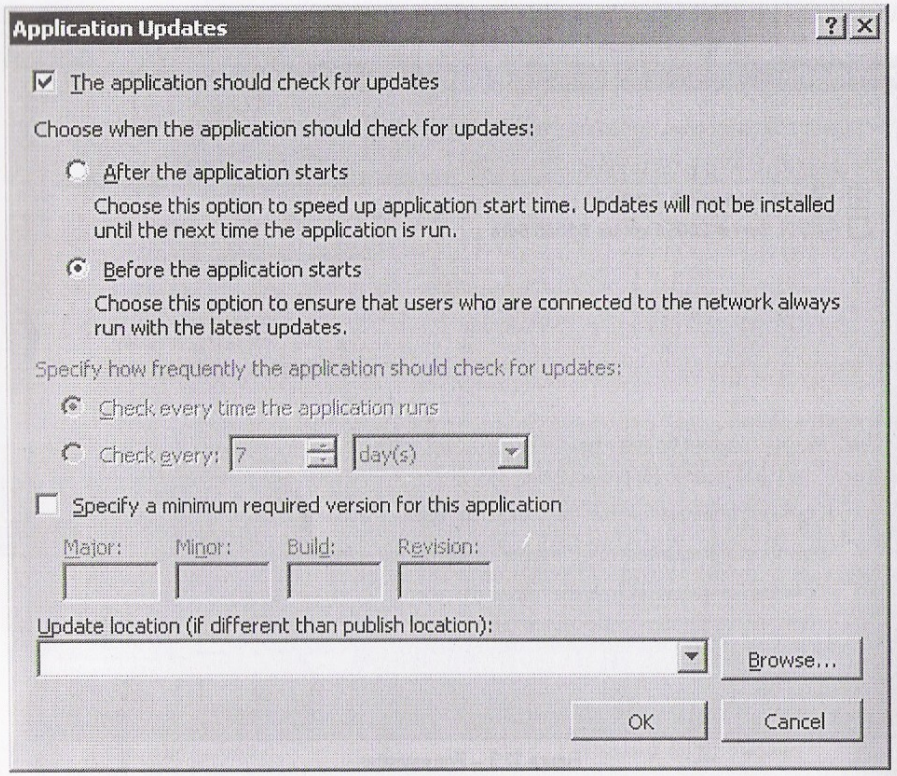


Figura 12.4 – *Updates*

Esta tela contém toda a configuração de controle de atualizações com possibilidade de agendamento, versão mínima para executar e local da fonte caso tenha uma nova versão. Algo semelhante ao *Windows Update*.

Clique em OK e selecione *Options* (Figura 12.5).

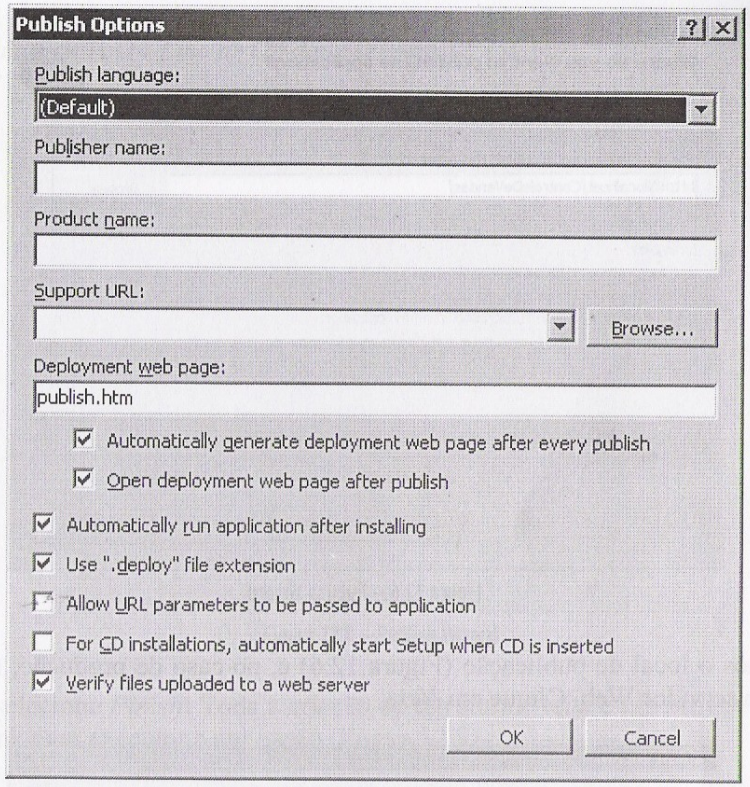


Figura 12.5 – *Options*

Esta tela apresenta algumas configurações mais simples como: nome do produto, extensão, instalação em CD etc.

Prática

Para utilizar todos esses recursos é muito simples: clique em *Publish Wizard* e uma janela é aberta (Figura 12.6):

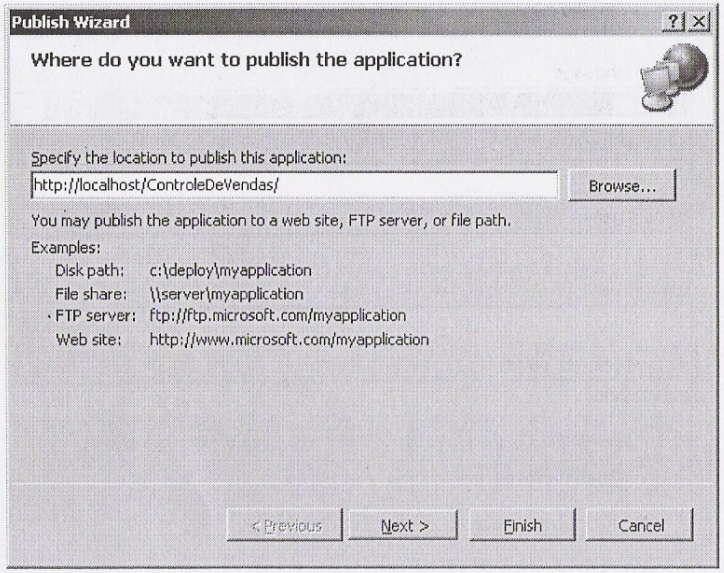


Figura 12.6 – Publish Wizard

Definimos o local de publicação (Figura 12.6) e, no caso de produção, selecionaríamos o servidor Web. Clique em *Next*.

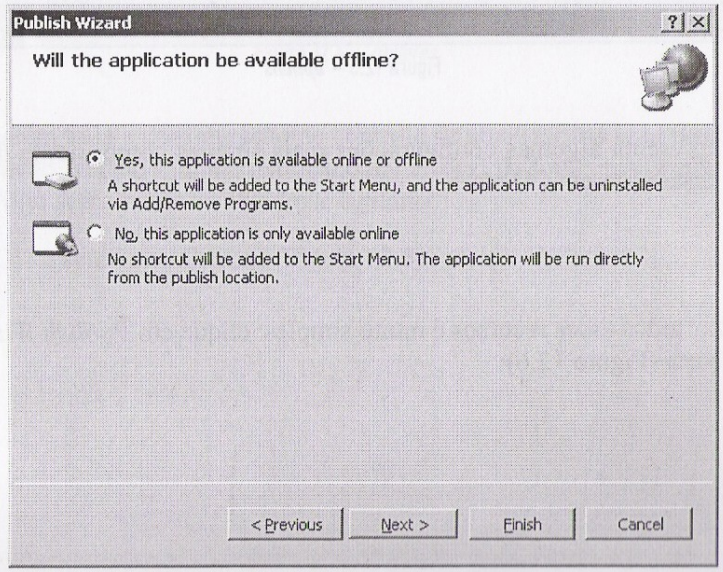


Figura 12.7 – Publish Wizard

Temos a possibilidade de suporte On-Line ou Off-Line da aplicação; selecione *Yes, this application is On-Line or Off-Line* (Figura 12.7).

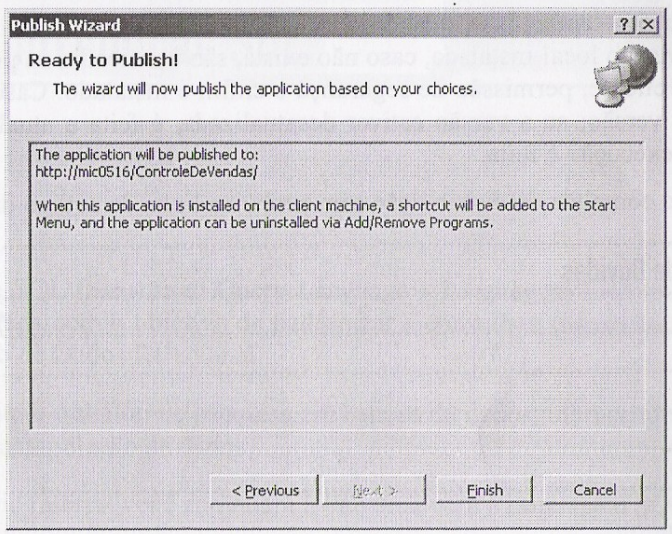


Figura 12.8 – Publish Wizard

Pronto, selecione *Finish*! Toda a criação da estrutura é feita e um browser é aberto com o link para executar a aplicação (Figura 12.9).

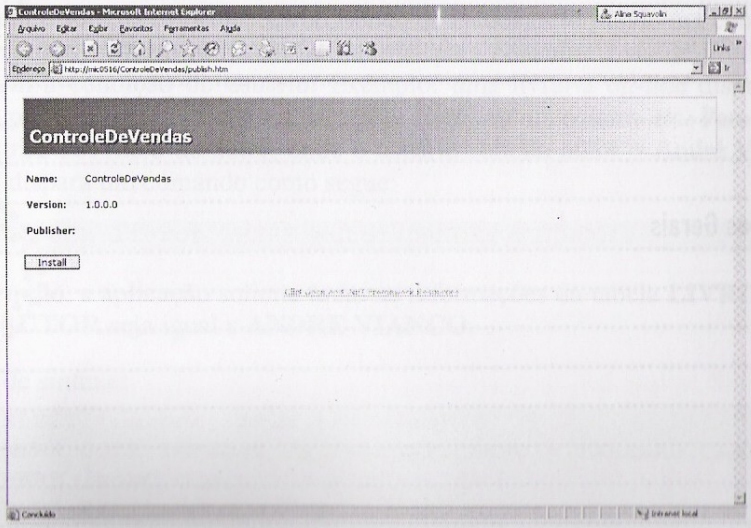


Figura 12.9 – Página do template

Clique em *Install*, uma mensagem de segurança é aberta, clique em *Install* e pronto. Após toda a verificação, o formulário é exibido.

Para facilitar o entendimento, quando clicamos no botão *Install* é verificado se já existe uma versão local instalada, caso não exista, são verificadas as pendências na máquina do cliente, permissão de segurança e enfim é instalado. Caso já exista, é verificada a versão; se a versão estiver desatualizada, é feita a atualização; caso contrário, a execução é feita.

Uma solução completa de distribuição sem nenhuma linha de código digitada.

Anotações de Dúvidas

Preciso Revisar

Anotações Gerais

Introdução SQL

SQL

A linguagem SQL (Structured Query Language – Linguagem Estruturada de Pesquisa) foi criada com o objetivo de padronizar a consulta e manipulação de dados em um Banco de Dados Relacional.

A SQL facilita o trabalho de pesquisa em bancos de dados, oferecendo dezenas de recursos de manipulação de dados.

Neste pequeno tutorial você aprenderá a trabalhar com a linguagem SQL

Principais comandos

SELECT

Este comando solicita informações ao Banco de Dados, retornando um conjunto de registros (linhas). Muito utilizado na Internet para consultar lista de produtos, de acordo com a condição do usuário. Exemplo: uma livraria on-line disponibiliza consultas por diferentes campos, tais como: livro, autor, editora etc. Para realizar uma pesquisa por meio do autor, onde o usuário solicita autor = André Vianco, a aplicação dispara um comando como segue:

```
SELECT * FROM livros WHERE autor="ANDRÉ VIANCO",
```

Neste exemplo, a aplicação solicita todas as informações da tabela **LIVROS**, onde o campo **AUTOR** seja igual a **ANDRE VIANCO.**

Exemplo de sintaxe:

```
SELECT [campos] FROM [tabelas]
INNER JOIN TABELA2 ON TABELA1.CAMPO = TABELA2.CAMPO
WHERE [condição]
ORDER BY [campos de ordenação]
GROUP BY [campos de agrupamento]
```

SELECT	Selecionar registros
FROM	Lista de tabelas dos campos referidos em SELECT e considerados em WHERE
Inner Join	Relacionamento entre tabelas. A claúsula Inner Join combina registros de duas tabelas sempre que campos comuns satisfaçam um valor de comparação
Left Join	Seleciona todos os registros da primeira tabela (à esquerda da seleção), mesmo que não haja registros correspondentes na segunda tabela
Right Join	Seleciona todos os registros da segunda tabela (à direita da seleção), mesmo que não haja registros correspondentes na segunda tabela
WHERE	Especifica as restrições que afetam as linhas a serem selecionadas, bem como define a condição de junção entre as tabelas envolvidas
GROUP BY	Define as colunas que fazem parte da condição de agrupamento. Todas as colunas requeridas em SELECT, exceto as que representam funções, devem ser indicadas.
ORDER BY	Define quais colunas farão parte da condição de ordenação. A nomenclatura ASC (default) define ordenação ascendente ou DESC para descendente.

COMANDO	DESCRIÇÃO
campos	Nomes dos campos a serem retornados. Caso um asterisco (*) seja especificado, todos os campos das tabelas são retornados.
Tabelas	A(s) tabela(s) de onde serão extraídos os dados.
Condição	Condição para o retorno das informações. A condição pode usar os operadores booleanos como <, >, <>, =, AND, OR, NOT.
campos de ordenação	As informações serão organizadas através do(s) campo(s) utilizados nesta cláusula.
campos de agrupamento	Os campos usados para agrupar campos quando são usadas funções de agregação como COUNT, SUM e AVG.

Exemplos de utilização:

☑ Retorna todos os valores de **todos** os campos da tabela **CLIENTE.**

```
SELECT * FROM CLIENTE
```


☑ Retorna todos os valores dos campos **cli_Id** e **cli_NomeRazao** da tabela **CLIENTE**

```
SELECT cli_Id,
   cli_NomeRazao
FROM CLIENTE
```

☑ Retorna todos os valores do campo **cli_NomeRazao** da tabela **CLIENTE** onde o campo **cli_UF** = "SP".

```
SELECT cli_NomeRazao
 FROM CLIENTE
 WHERE cli_UF = "SP"
```

☑ Retorna todos os valores do campo pro_Descricao da tabela PRODUTO onde o campo pro_Valor esteja entre R$ 20,00 e R$ 30,00

```
SELECT pro_Descricao
 FROM PRODUTO
 WHERE pro_Valor BETWEEN 20 and 30
```

☑ Retorna todos os valores dos campos pro_Descricao e pro_Valor da tabela PRODUTO onde o campo pro_Valor esteja entre R$ 20,00 e R$ 30,00, ordenando o resultado por valores.

```
SELECT pro_Descricao, pro_Valor
 FROM PRODUTO
 WHERE pro_Valor BETWEEN 20 and 30
 ORDER BY pro_Valor
```

A cláusula SELECT nos deixa muito à vontade para a formação de consultas.

Usando LIKE e caracteres "curinga"

O comando LIKE é usado em conjunto com os caracteres % e _ . O caractere % pode ser comparado ao "*" (asterisco) do DOS, e o caractere _ (*underscore*) é semelhante ao caractere (?) também do DOS.

Exemplos:

☑ Retorna todos os valores do campo cli_NomeRazao onde o campo em questão começa com "ACME"

```
SELECT cli_NomeRazao
 FROM CLIENTE
 WHERE cli_NomeRazao Like 'ACME%'
```

Vale a pena salientar que, na utilização do LIKE, se utilizarmos o curinga antes da palavra chave, perdemos o ganho de desempenho que temos na utilização de índices. Veremos índices logo mais adiante.

Funções Especiais

FUNÇÃO	TIPO	DESCRIÇÃO
AVG()	Agregação	Calcula a média dos valores de uma coluna
COUNT()	Agregação	Retorna o número de linhas resultante de uma Query
MAX()	Agregação	Retorna o maior valor de uma coluna
MIN()	Agregação	Retorna o menor valor de uma coluna
SUM()	Agregação	Totaliza os valores de uma coluna

Exemplos:

☑ Retorna o total de clientes por UF, ordenando alfabeticamente o campo cli_UF.

```
SELECT cli_UF, count(*) FROM CLIENTE
 GROUP BY cli_UF
 ORDER BY cli_UF
```

☑ Retorna a média de estoque dos produtos

```
SELECT AVG(pro_QtdeEstoque) FROM PRODUTO
```

☑ Retorna o valor total do estoque físico

```
SELECT SUM(pro_QtdeEstoque * pro_Valor) FROM PRODUTO
```

☑ Retorna o menor valor de um PEDIDO

```
SELECT MIN(ped_Valor) FROM PEDIDO
```

☑ Retorna o maior valor de um PEDIDO

```
SELECT MIN(ped_Valor) FROM PEDIDO
```

INSERT

O comando INSERT é utilizado para inserir registros em tabelas.

Sintaxe.

```
INSERT INTO Tabela (Campo1, Campo2, ...) VALUES (Valor1,
Valor2, ...)
```


Vejamos alguns exemplos:

COMANDO	DESCRIÇÃO
`INSERT INTO Produto (pro_Descricao, pro_QtdeEstoque) VALUES ('VS 2005 Professional', 22000)`	Insere um novo registro na tabela Produto, com os dados (pro_Descricao = "VS 2005 Professional" e pro_QtdeEstoque = 22000) Os demais campos da tabela não recebem valor, portanto o conteúdo é igual a nulo.
`INSERT INTO Produto VALUES (2005,'VS 2005 Professional', 22000, 3500)`	Insere um novo registro na tabela Produto, preenchendo todos os campos. Quando preenchemos todos os campos, não é necessário discriminá-los na lista de campos.

UPDATE

O comando UPDATE é utilizado para atualizar dados de uma tabela. Sintaxe básica:

```
UPDATE Nome da Tabela SET Campo = Valor WHERE Condição
```

Vejamos alguns exemplos:

COMANDO	EFEITO / COMENTÁRIO
`UPDATE PESSOAL SET Salario = 3500 WHERE DEPTO = 'DESENVOLVIMENTO'`	Atualiza a tabela PESSOAL, alterando o campo SALARIO para 3500, cujo campo DEPTO seja igual a "DESENVOLVIMENTO".
UPDATE PESSOAL SET Salario = Salario * 1.15	Atualiza a tabela PESSOAL, reajustando todos os salários em 15% (1.15).

É importante salientar que o comando ***Update*** na maioria das vezes é utilizado com a cláusula ***where.*** Se não houver tal condição, a tabela inteira é atualizada.

DELETE

O comando DELETE é utilizado para apagar (excluir) dados em uma tabela.

Sintaxe básica:

```
DELETE from Nome da Tabela WHERE Condição
```

Vejamos alguns exemplos:

COMANDO	EFEITO / COMENTÁRIO
DELETE FROM CLIENTE **WHERE** cli_id = 353	Exclui da tabela CLIENTE o registro cujo campo cli_id = 353
DELETE FROM CLIENTE	Exclui **todos** os registros da tabela **CLIENTE**

É importante salientar que o comando **Delete** na maioria das vezes é utilizado com a cláusula **where**. Se não houver tal condição, todos os registros da tabela são excluídos.

Referências Bibliográficas

Sites

www.aspnet.com
www.codificando.net
www.devmedia.com.br/msdn
www.juliobattisti.com.br
www.linhadecodigo.com.br
www.macoratti.net
www.msdnbrasil.com.br
www.msdn.com
www.projetando.net

Índice Remissivo

.

.NET, 3, 4, 8, 11

A

Access, 67
ADO.NET, 12
Asp.NET, 6
ASP.Net, 3
AVG, 168

B

banco de dados, 122
Banco de Dados, 67, 167
Base Class Library, 12

C

C#, 39
C++., 7
classes, 63
Classes Globais, 107
Click Once, 4
Code-Snippet, 8
Common Language Runtime, 11
Common Type System, 39
Consulta, 78
conversões de valores, 43
COUNT, 168

D

DataBound, 34
DataReader, 15
DataSet, 15
Diagrama de classes, 70
distribuição, 159

F

ferramentas Express, 5, 8, 67
For, 52
For Each/In, 53
Formulário, 75, 131, 139, 144, 150
Framework, 12, 13, 18
Framework .NET, 3, 10

G

GROUP BY, 167

I

IDE, 31
INSERT, 170, 171
Interface e Regra de Negócio, 130

L

laços, 52
LIKE, 169

M

Metadados, 11
MSDE, 4
MSIL, 11

O

ODBC, 13
OLE DB, 13
Oracle, 13
ORDER, 167

S

SELECT, 167, 168
Solution Explorer, 33
SQL, 122, 167, 171, 172
SQL Server, 4, 13
Start Page, 8
Starter Kits, 8
Strings, 45
SUM, 168
System.Exception, 56

T

Tomada de Decisão, 47, 50
Tratamento de Erros – Exception, 56

U

UPDATE, 171, 172

V

VB.NET, 18
Versões Express, 4
Visual Basic Express, 73, 108
Visual Basic.NET, C# e J#., 6
Visual Basic.NET., 6
Visual C#., 7
Visual Studio, 4, 8
Visual Studio Express, 17

W

WHERE, 167, 171, 172
While, 54
Windows Forms, 3

X

XML, 12, 109

Participe do **BRASPORT INFOCLUB**

Preencha esta ficha e envie pelo correio para a

BRASPORT LIVROS E MULTIMÍDIA

Rua Pardal Mallet, 23 – Cep.: 20270-280 – Rio de Janeiro – RJ

Você, como cliente BRASPORT, será automaticamente incluído na nossa Mala Direta, garantindo o recebimento regular de nossa programação editorial.
Além disso, você terá acesso a ofertas incríveis, exclusivas para os nossos leitores.
Não deixe de preencher esta ficha.
Aguarde as surpresas. Você vai sentir a diferença!

Nome: _____
Endereço residencial: _____
Cidade: _____ Estado: _____ Cep.: _____
Telefone residencial: _____
Empresa: _____
Cargo: _____
Endereço comercial: _____
Cidade: _____ Estado: _____ Cep.: _____
Telefone comercial: _____
E-mail: _____

Gostaria de receber informações sobre publicações nas seguintes áreas:

- ❒ linguagens de programação
- ❒ planilhas
- ❒ processadores de texto
- ❒ bancos de dados
- ❒ engenharia de software
- ❒ hardware
- ❒ redes
- ❒ editoração eletrônica
- ❒ computação gráfica
- ❒ multimídia
- ❒ internet
- ❒ saúde
- ❒ sistemas operacionais
- ❒ outros _____

Comentários sobre o livro _____