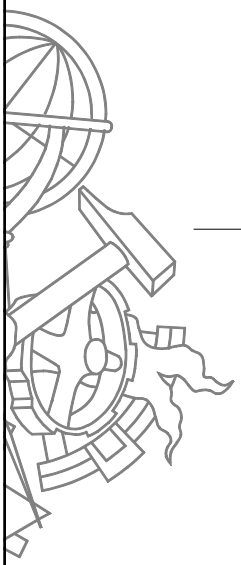


Ambientes de Desenvolvimento Avançados

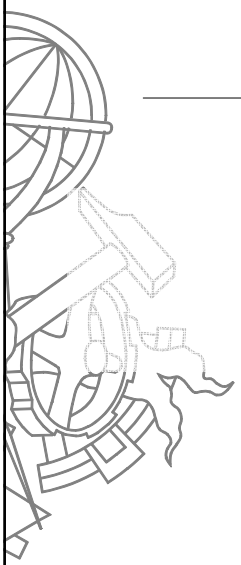
http://www.dei.isep.ipp.pt/~jtavares/ADAV

Aula Teórico-Prática

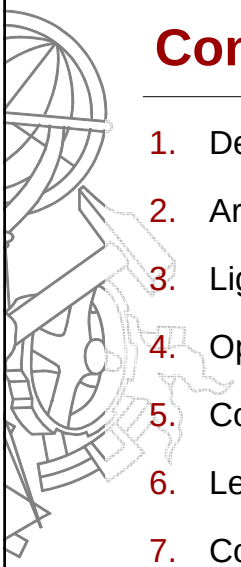
Engenharia Informática

2005/2006

José António Tavares
jrt@isep.ipp.pt

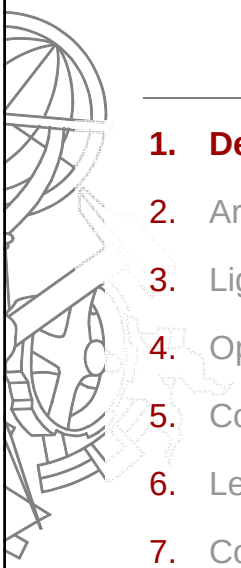


Programação com ADO.NET



Conteúdo

1. Desenho de aplicações centradas em dados
2. Arquitectura ADO.NET
3. Ligação a Base de Dados
4. Operações ligadas à Base de Dados
5. Construção de *Data Sets*
6. Ler e Escrever XML com ADO.NET
7. Construir *Data Sets* a partir de Bases de Dados



1. **Desenho de aplicações centradas em dados**
2. Arquitectura ADO.NET
3. Ligação a Base de Dados
4. Operações ligadas à Base de Dados
5. Construção de *Data Sets*
6. Ler e Escrever XML com ADO.NET
7. Construir *Data Sets* a partir de Bases de Dados

Cenários Conectados

- Um cenário conectado é aquele no qual os utilizadores estão permanentemente ligados à bases de dados

- Vantagens:
  - É mais fácil exercer segurança ao nível do ambiente
  - A concorrência é mais fácil de controlar
  - Os dados estão mais actualizados que nos outros cenários
- Desvantagens
  - É necessário haver uma ligação constante ao servidor
  - Escalabilidade

Cenários Desconectados

- Num ambiente desconectado, um sub-conjunto de dados pode ser copiado e modificado independentemente e mais tarde as alterações podem ser introduzidas de novo na base de dados
- Vantagens
  - Pode-se trabalhar a qualquer altura e pode-se efectuar uma ligação à base de dados apenas quando necessário
  - Outros utilizadores podem usar os recursos
  - Este tipo de ambientes aumenta a escalabilidade e desempenho das aplicações
- Desvantagens
  - Os dados nem sempre estão actualizados
  - Podem ocorrer conflitos de dados que têm que ser resolvidos

O que é o ADO.NET?

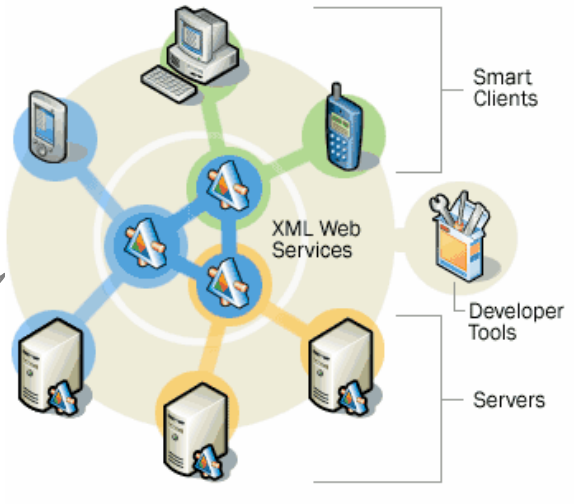
- ADO.NET é a solução de acesso a dados em .NET
  - Faz parte integrante *Framework* .NET
  - Substitui completamente o ADO 'clássico'
- Constitui uma evolução ao 'clássico' ADO
  - Existem similaridades mas mudou muito
- Suporta os modelos conectados e desconectados
  - DataReaders para acesso conectados a dados
  - DataSets e DataAdapters para desconectados
- Baseado em "*pluggable* Data Providers"
  - SQL Server, Oracle, OLE DB & ODBC *standard*

ADO.NET

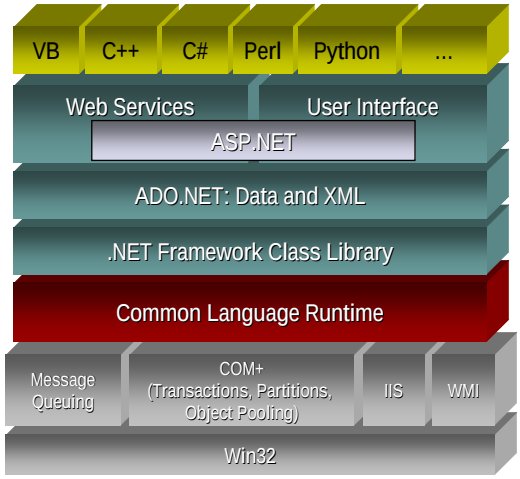
A nível do armazenamento de Dados, o ADO.NET suporta vários tipos:

- Não estruturados;
- Estruturados, não-hierárquicos
  - Ficheiros CSV (Comma Separated Value), Folhas Microsoft Excel, Ficheiros Microsoft Exchange, …
- Hierárquicos
  - Documentos XML e outros
- Bases de Dados Relacionais
  - SQL Server, Oracle, Access, ODBC, …

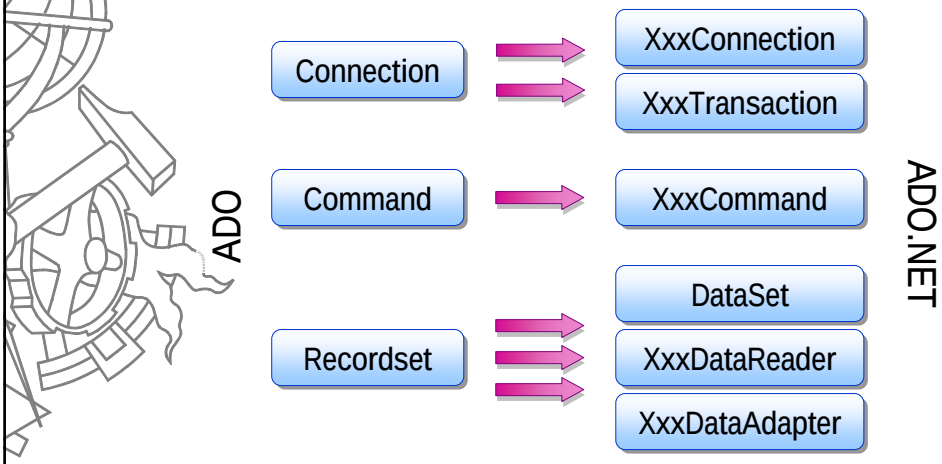
Visão .net



ADO.NET e a *Framework* .NET



Evolução do ADO para ADO.NET



ADO vs. ADO.NET 1/2

ADO

- Desenhado para acessos conectados
- Amarrado ao modelo físico dos dados
- **RecordSet** é o contentor central de dados
- **RecordSet** é uma (1) tabela que contem todos os dados
  - Obter dados de > 1 tabela requer um comando JOIN
  - Os dados "são planos": perde relações e a navegação é sequencial
- Os tipos de dados estão limitados aos tipos de dados COM/COM+
- A partilha de dados é via COM *marshalling*
- Problemas com *marshalling* através *firewalls* (DCOM, binários)

ADO vs. ADO.NET 2/2

ADO.NET

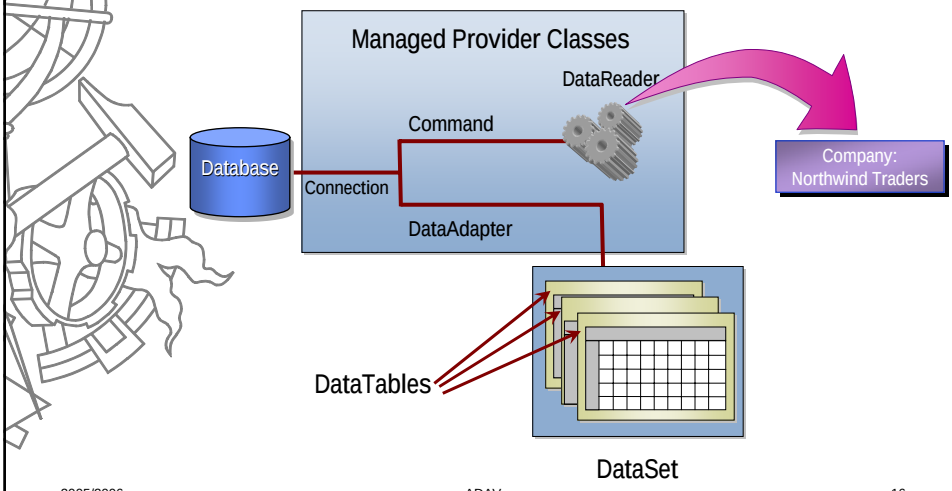
- Desenhado acessos desconectados
- Pode modelar dados logicamente (em memória)!
- **DataSet** substitui o **RecordSet**
- **DataSet** pode conter múltiplas tabelas
  - Obter dados de > 1 tabela não requer um comando JOIN
  - Relações são preservadas e a navegação é relacional
- Os tipos de dados estão apenas limitados por esquemas (*schema*) XML
- Não é necessário conversão de tipo de dados
- XML, tal como HTML, é texto: "*Firewall friendly*"

Beneficios do ADO.NET

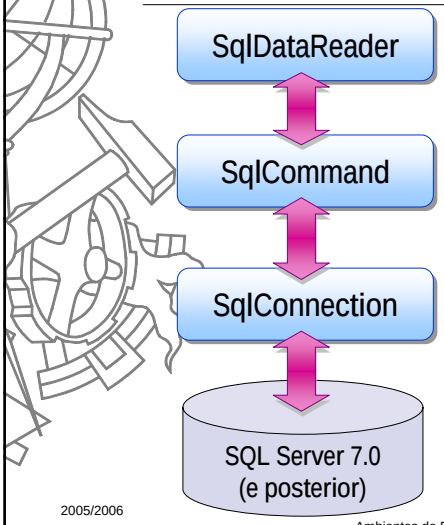
- Interoperacionalidade através do uso de XML
  - *Standard* aberto
  - Texto legível e compreensível pelos humanos
  - Dados descrevem-se a si mesmos (XML)
  - Usado transferir todos os dados em ADO.NET
- Escalabilidade através de **DataSet** desconectados
  - Conecções não são mantidas por longos períodos
  - "*Database locking*" não existe
  - Works the way the Web works: "Hit and Run!"
- "*Maintainability*"
  - Separação entre o tratamento de dados e a "*user interface*"

1. Desenho de aplicações centradas em dados
2. **Arquitectura ADO.NET**
3. Ligação a Base de Dados
4. Operações ligadas à Base de Dados
5. Construção de *Data Sets*
6. Ler e Escrever XML com ADO.NET
7. Construir *Data Sets* a partir de Bases de Dados

Arquitectura do ADO.NET



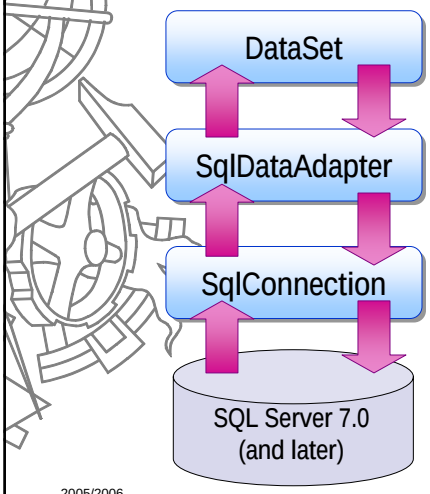
Usar as classes ADO.NET num cenário conectado



Num cenário conectado, os recursos são mantidos no servidor até a ligação ser fechada

1. Abrir ligação
2. Executar comando
3. Processar linhas no *reader*
4. Fechar *reader*
5. Fechar ligação

Usar as classes ADO.NET num cenário desconectado



Num cenário desconectado, os recursos não são mantidos no servidor durante o processamento dos dados

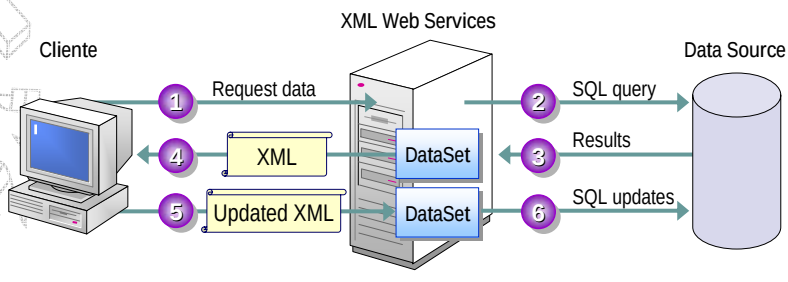
1. Abrir a ligação
2. Encher o DataSet
3. Fechar a ligação
4. Processar o DataSet
5. Abrir a ligação
6. Actualizar a fonte de dados
7. Fechar a ligação

Arquitectura do ADO.NET

Esta animação descreve a arquitectura do ADO.NET considerando ambos os modelos: conectado e desconectado

ADO.NET e XML

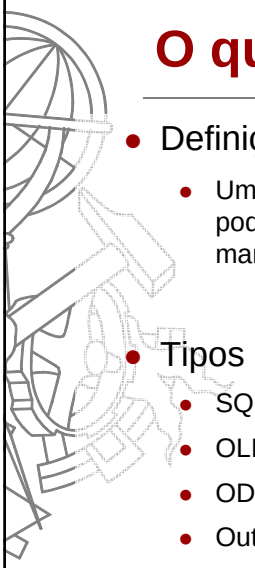
O ADO.NET é totalmente integrado com XML



Utilização do XML numa aplicação ADO .NET desconectada

Programação com ADO.NET

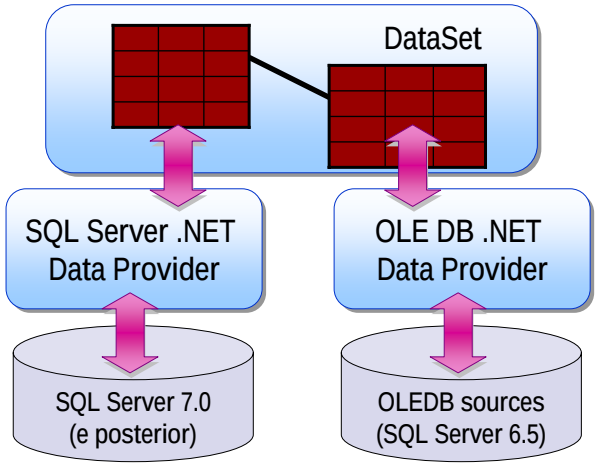
1. Desenho de aplicações centradas em dados
2. Arquitectura ADO.NET
3. **Ligação a Base de Dados**
4. Operações ligadas à Base de Dados
5. Construção de *Data Sets*
6. Ler e Escrever XML com ADO.NET
7. Construir *Data Sets* a partir de Bases de Dados



O que são .NET *Data Providers*

- Definição
  - Um .NET "*data provider*" é um conjunto de classes que pode ser usado para efectuar ligações a bases de dados, manipular e actualizar os dados
- Tipos de .NET *data providers*
  - SQL Server .NET Data Provider
  - OLE DB .NET Data Provider
  - ODBC .NET Data Provider
  - Outros (DB2/400, MySQL, …)

Modelo de objectos do ADO.NET



Os Namespaces

System.Data

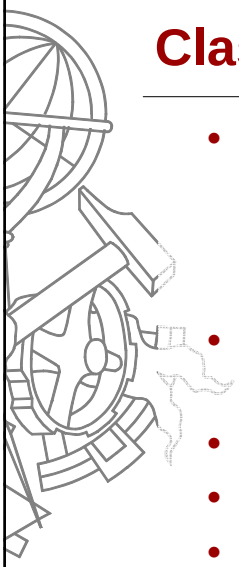
System.Data.Common

System.Data.SqlClient

System.Data.OleDb

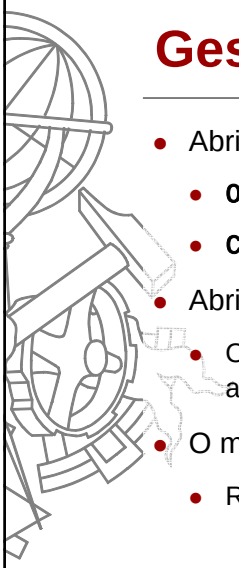
System.Data.SqlTypes

System.Xml



Classes do .NET *Data Provider*

- XxxConnection – *exemplo, SqlConnection*
  - XxxTransaction – *exemplo, SqlTransaction*
  - XxxException – *exemplo, SqlException*
  - XxxError – *exemplo, SqlError*
- XxxCommand – *exemplo, SqlCommand*
  - XxxParameter – *exemplo, SqlParameter*
- XxxDataReader – *exemplo, SqlDataReader*
- XxxDataAdapter – *exemplo, SqlDataAdapter*
- XxxPermission – *exemplo, SqlClientPermission*



Gestão de conecções

- Abrir e fechar conexões explicitamente:
  - **`Open`**
  - **`Close`**
- Abrir e fechar ligações implicitamente:
  - Os "***Data Adapters***" podem abrir e fechar as ligações automaticamente sempre que necessário
- O método **`Dispose`**:
  - Remove a conexão da "*pool*" de conexões

OleDbConnection e SQLConnection

- Criação, abertura e fecho de uma conecção a uma fonte de dados
- Exemplo **OleDbConnection** :

```
String conStr="Provider=Microsoft.Jet.OLEDB.4.0;" +
              "Data Source=NWIND_RW.MDB";
OleDbConnection aConn = new OleDbConnection(conStr);
aConn.Open();
// acesso, inserção e actualização de dados
aConn.Close();
```


Pooling de Conecções

- A abordagem desconectada melhora significativamente a escalabilidade
  - No entanto, requisitar frequentemente conecções degrada o desempenho das aplicações
- O "*pooling*" de conecções permite uma '*pool*' de conecções idênticas partilhadas por múltiplos utilizadores/aplicações
  - Escalável e rápido
  - Os maiores benefícios são para as aplicações Web / n-tier
- ADO.NET usa "*pooling*" por defeito

Pooling de Conecções com o SQL Server

Esta animação descreve como é que o *pooling* de conecções funciona com o Microsoft SQL Server 2000

1. Desenho de aplicações centradas em dados
2. Arquitectura ADO.NET
3. Ligação a Base de Dados
4. **Operações ligadas à Base de Dados**
5. Construção de *Data Sets*
6. Ler e Escrever XML com ADO.NET
7. Construir *Data Sets* a partir de Bases de Dados

O objecto COMMAND

- O objecto *command* é uma referência a uma instrução de SQL ou *Stored Procedure*

- Propriedades
  - (Name), Connection, CommandType, CommandText, Parameters

- Métodos
  - ExecuteScalar, ExecuteReader, ExecuteNonQuery
  - ExecuteXmlReader (só para o SqlCommand)

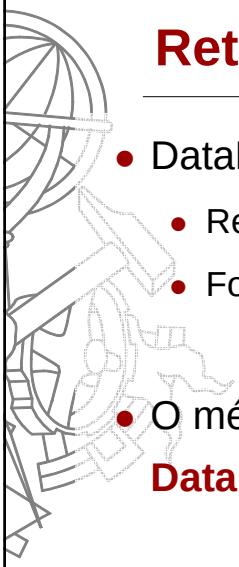
Retornar só um registo

O ADO.NET é mais eficiente que o ADO "clássico", dado que este retorna um *RecordSet* completo

Exemplos de utilização:

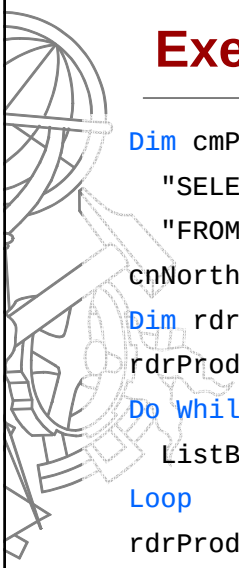
- Obter as unidades em stock para um determinado produto
- Contar os produtos
- COUNT, MAX, MIN, AVERAGE

Exemplo



Retornar múltiplos registos

- DataReader
  - Read-only
  - Forward-only
- O método ExecuteReader retorna um **DataReader**



Exemplo Data Reader

```
Dim cmProducts As New SqlCommand( _
  "SELECT ProductName, UnitsInStock " & _
  "FROM Products", cnNorthwind)
cnNorthwind.Open()
Dim rdrProducts As SqlDataReader
rdrProducts = cmProducts.ExecuteReader()
Do While rdrProducts.Read()
  ListBox1.Items.Add(rdrProducts.GetString(0))
Loop
rdrProducts.Close()
```

Instruções de Inserção e Actualização de Dados

```
Dim cmd As SqlCommand = New SqlCommand(_
   "dbo.IncreaseProductPrices", cnNorthwind)

cmd.CommandType = CommandType.StoredProcedure
cnNorthwind.Open()

Dim affected As Integer = cmd.ExecuteNonQuery()

cnNorthwind.Close()
MessageBox.Show(“Registos Afectados: " & affected)
```


Transacções

- XxxConnection – por exemplo, SqlConnection
  - BeginTransaction
- XxxTransaction – por exemplo, SqlTransaction
  - Commit
  - Rollback

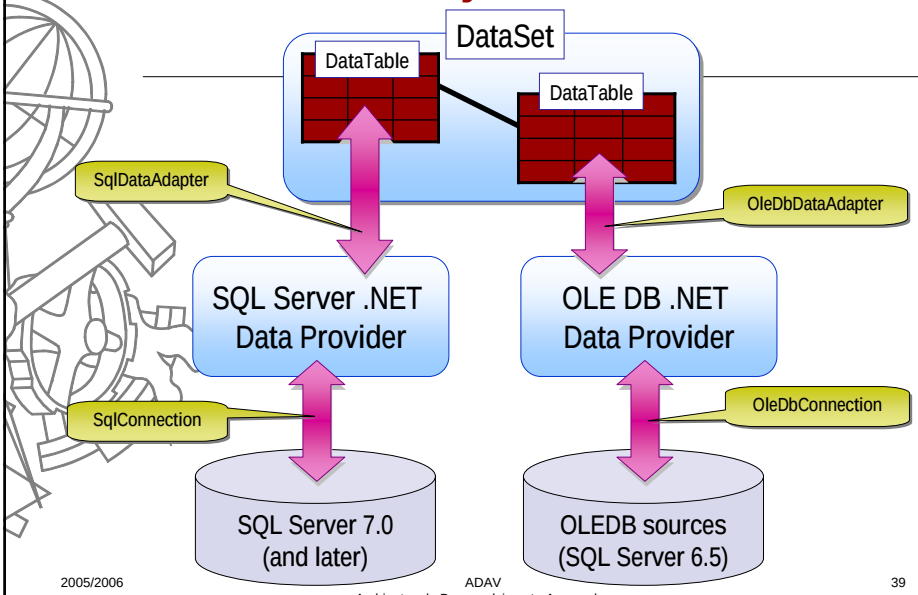
Exemplo

DEMO 01

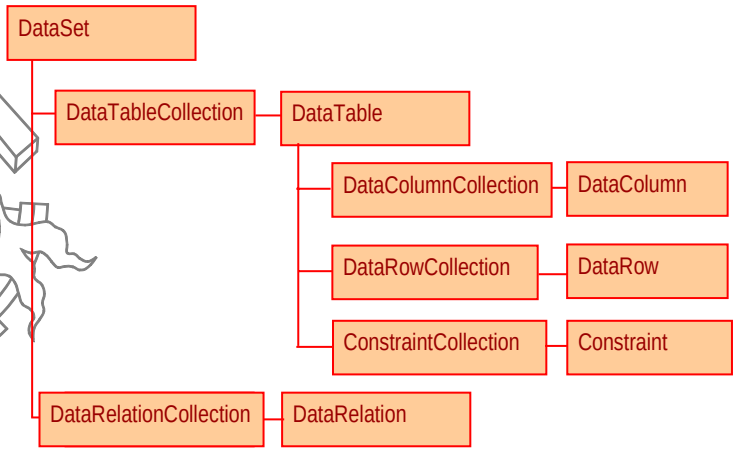
- Criar uma solução em Visual Studio .NET do tipo *Database*
- Criar uma pequena aplicação que ilustra o modelo conectado

1. Desenho de aplicações centradas em dados
2. Arquitectura ADO.NET
3. Ligação a Base de Dados
4. Operações ligadas à Base de Dados
5. **Construção de *Data Sets***
6. Ler e Escrever XML com ADO.NET
7. Construir *Data Sets* a partir de Bases de Dados

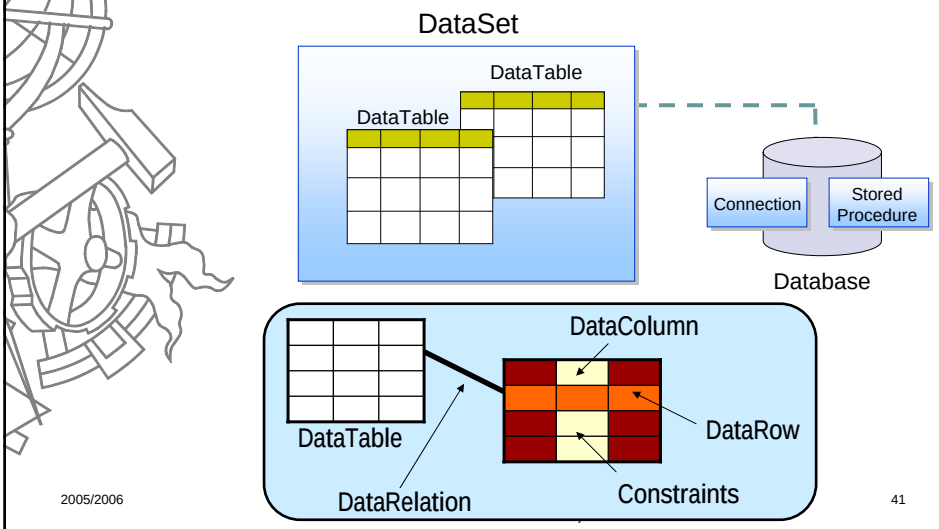
Modelo de Objectos do ADO.NET



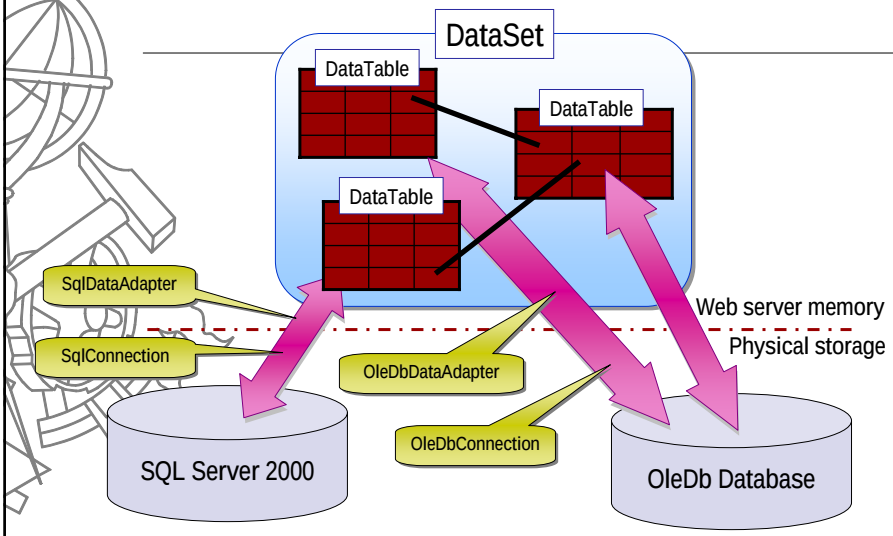
Modelo de Objectos do DataSet



Como é constituído o DataSet



O que é DataSet?



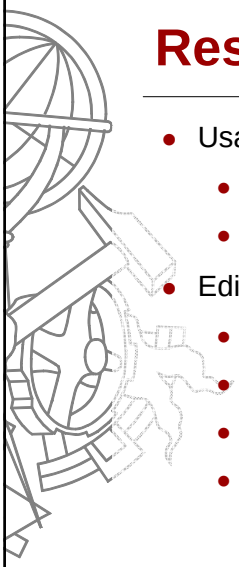
DataSet, DataTable, DataColumn

- **Criar um DataSet**
  - Arrastar o controlo DataSet da Toolbox
- **Criar uma DataTable**
  - Editar a colecção de Tables de um DataSet usando a janela de propriedades
- **Criar uma DataColumn e adicioná-la a uma DataTable**
  - Editar a colecção de Columns de uma DataTable usando a janela de propriedades

System.Data — DataSet e DataTable

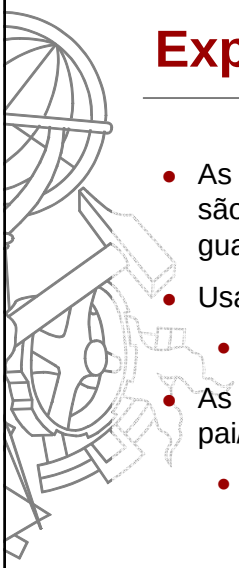
Criar **DataTable** e adicionar a um **DataSet**

```
DataSet ds = new DataSet();

// Create DataTable object: “Customers”.
DataTable dt= new DataTable( “Customers” );

// Create and add columns to the table
// 1. Explicitly create and Add a DataColumn
DataColumn dc = new DataColumn( “CustID”, Int16 );
dt.Columns.Add( dc );

// 2. Implicitly Create and Add columns (DataColumn).
dt.Columns.Add( “First_Name”, String );
dt.Columns.Add( “Last_Name”, String );

// Add the DataTable object to the DataSet
ds.Tables.Add( dt );
```



Restrição de Chave Primária

- Usar a propriedade PrimaryKey da DataTable
  - Seleccionar as colunas pela ordem desejada
  - Não permite mudar o nome à restrição
- Editar a colecção de Constraints da DataTable
  - Adicionar uma UniqueConstraint
  - Dar o nome à restrição
  - Seleccionar as colunas
  - Activar a caixa de chave primária



Expressões “Customisadas”

- As expressões “*Custom*” são colunas cujos valores são originados em cálculos e não em valores guardados
- Usar a propriedade Expression da DataColumn
  - `Sum([Unit Price] * [Quantity])`
- As funções de agregação podem usar relações pai/filho
  - `Avg, Count, Sum, Max, Min`

DEMO 02

- Um pequena aplicação que ilustra programaticamente, no modelo desconectado, a criação de *DataSets*

Inserir registos

- Criar uma linha nova

```
DataRow drNovoEmpregado = dtEmpregados.NewRow();
```

- Preencher a nova linha

```
drNovoEmpregado["EmpregadoID"] = 11;
drNovoEmpregado["Nome"] = "Nuno";
```

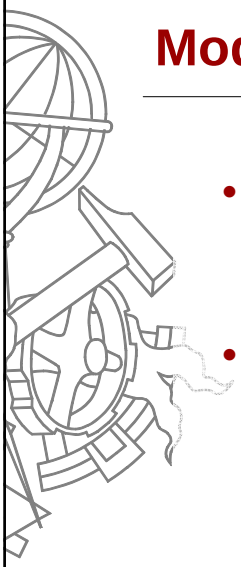
- Adicionar a linha a uma DataTable

```
dtEmpregados.Rows.Add(drNovoEmpregado);
```

- Criar, preencher e adicionar numa só instrução

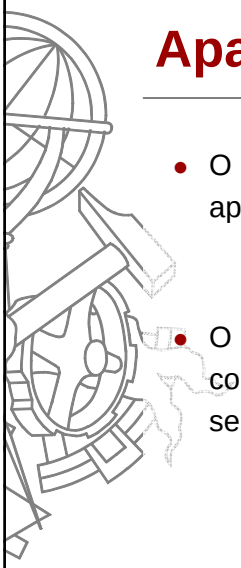
```
dtEmpregados.Rows.Add(new Object[]{11, "Nuno"});
```

Modificar registos



- O método **BeginEdit** da classe DataRow
  - Desliga o lançamento de eventos e excepções
- Os métodos **EndEdit** e **CancelEdit** da classe **DataRow**
  - Ligam de novo o lançamento de eventos e excepções

Apagar registos



- O método **Remove** da classe **DataRowCollection** apaga completamente o registo da colecção

```
dtEmpregados.Rows.Remove(drEmpregado);
```

- O método **Delete** da classe **DataRow** marca o registo como apagado. Este fica escondido, mas acessível, se necessário

```
drEmpregado.Delete();
```

Alterações de dados no DataSet

Uma **DataRow** pode guardar múltiplas *versões* de valores para cada coluna:

- DataRowVersion.Current
  - O valor actual da coluna
- DataRowVersion.Original
  - O valor da coluna antes de qualquer alteração ter ocorrido
- DataRowVersion.Proposed
  - O valor da coluna durante o ciclo **BeginEdit** / **EndEdit**
- DataRowVersion.Default
  - O valor por defeito em função do "row's *state*"

Processamento de Versões de linha

	CURRENT	ORIGINAL	PROPOSED
	White	**White**	**N/A**

```
dataRow.BeginEdit();
dataRow[ "au_lname" ] = "Brown";
```

	White	**White**	**Brown**

```
dataRow.EndEdit();
```

	Brown	**White**	**N/A**

```
dataRow[ "au_lname", DataRowVersion.Current ] // Brown
dataRow[ "au_lname", DataRowVersion.Original ] // White
```

Processamento do *Row State*

	CURRENT	ORIGINAL	ROW STATE
	White	**White**	**Unchanged**

```
dr["au_lname"] = "Brown";
```

	Brown	**White**	**Modified**

```
dr.AcceptChanges();
```

	Brown	**Brown**	**Unchanged**

Actualizações com o *DataAdapter*

Actualizações são escritas na base de dados usando o método **Update** do **DataAdapter**

- DataAdapter analisa o "*Row State*" para cada linha
- Executa a acção apropriada (*insert*, *update* or *delete*) de acordo com o seu "*row state*"
  - Usa os objectos **Command** atribuidos às suas propriedades **InsertCommand**, **UpdateCommand** e **DeleteCommand**

DataRows in DataTable	DataAdapter Action
RowState = Unchanged	Ignore
RowState = Added	Use INSERT command
RowState = Modified	Use UPDATE command
RowState = Deleted	Use DELETE command

Optimização de actualizações

- Objectos do tipo **DataSet** e **DataTable** suportam o método **GetChanges**
  - **GetChanges** sem argumentos
    - Extrai todas as linhas cujo **RowState** não é **Unchanged**

```
dsChanges = ds.GetChanges();
```

  - **GetChanges** com argumento do tipo "**RowState**"
    - Extrai apenas as linhas com um RowState especificado
    - Permite controlar a ordem em que os "*inserts*", "*updates*" e "*deletes*" são efectuados à base de dados

```
changes = ds.GetChanges( DataRowState.Added );
```


Usar o método SELECT

- As *DataTables* têm um método de ***Select*** que permite filtrar as *DataRows* de modo a cumprir com restrições de ordenação e de estado
- Três parâmetros opcionais
  - Expressões de filtragem, como por exemplo, "City='Porto'"
  - Ordenar, por exemplo, "City ASC"
  - DataViewRowState, por exemplo, **Deleted**

Criar um DataView

- Criar um DataView através da utilização dos controlos
- Criar um DataView programaticamente

```
DataView dvProdutos = new
      DataView(dsNorthwind.Tables["Products"]);
dvProdutos.Sort = "UnitPrice";
dvProdutos.RowFilter = "CategoryID > 4";
grdProducts.DataSource = dvProdutos;
```

- Aplicar um DataView a um DataTable

```
dvProdutos.Table = dsNorthwind.Tables("Products");
```


Objectos DataView - Exemplo

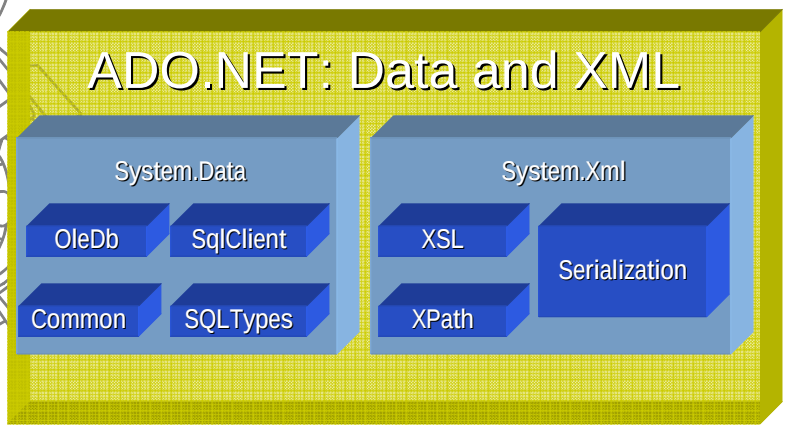
```
// Code for myTable “Customers” with
// “Name” column not shown
DataView view1 = new DataView( myTable );
DataView view2 = new DataView( myTable );

// Creates Ascending view of Customers by “Name”
view1.Sort = “Name ASC”;

// Set the view to show only modified (original) rows
view2.RowStateFilter=
            DataViewRowState.ModifiedOriginal;

// Bind to UI element(s)...
DataGrid myGrid = new DataGrid();
myGrid.SetDataBinding( view1, “Customer”);

//...
```

DEMO 03

- Considerando a aplicação do DEMO 02 e programaticamente:
  - Adicionar e actualizar registos
  - Criar *DataViews*

1. Desenho de aplicações centradas em dados
2. Arquitectura ADO.NET
3. Ligação a Base de Dados
4. Operações ligadas à Base de Dados
5. Construção de Data Sets
6. **Ler e Escrever XML com ADO.NET**
7. Construir *Data Sets* a partir de Bases de Dados

ADO.NET: Data e XML



XML e ADO .NET

- Método ReadXML

```
dsNorthwind.ReadXml(@"c:\XML\teste.xml",
  XmlReadMode.InferSchema);
```

- Método WriteXML

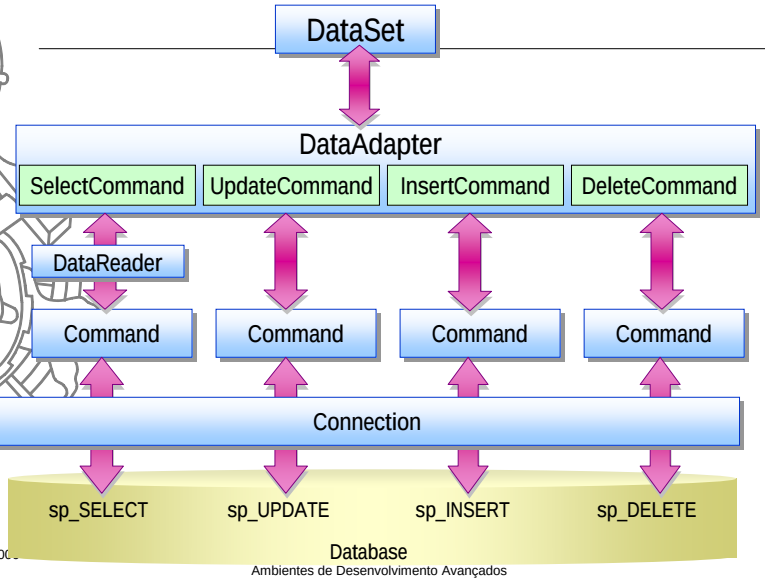
```
dsNorthwind.WriteXml(@"c:\XML\teste.xml",
  XmlWriteMode.WriteSchema);
```

DEMO 04

- Utilização de XML
- Um pequena aplicação que ilustra a persistência de *DataSets*

1. Desenho de aplicações centradas em dados
2. Arquitectura ADO.NET
3. Ligação a Base de Dados
4. Operações ligadas à Base de Dados
5. Construção de Data Sets
6. Ler e Escrever XML com ADO.NET
7. **Construir *Data Sets* a partir de Bases de Dados**

Modelo de Objectos do DataAdapter



DataAdapter

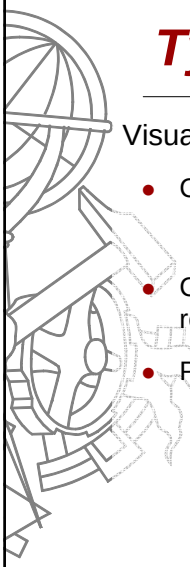
- Pode-se preencher um **DataSet** através de um **DataAdapter**
  - Invoca-se o método **Fill** do **DataAdapter**
- O método **Fill** executa o **SelectCommand**
  - Preenche o **DataSet** com a estrutura e conteúdo do resultado do inquérito
- Para optimizar o desempenho
  - `aDataSet.EnforceConstraints=False`
  - Chamar o método **BeginLoadData** do **DataTable**

DataAdapter - Exemplo

```
// Visual C#
DataSet dsCustomers = new DataSet();
dsCustomers.Tables.Add(new DataTable("Customers"));

dsCustomers.Tables[0].BeginLoadData();
daCustomers.Fill(dsCustomers, "Customers");
dsCustomers.Tables[0].EndLoadData();

dataGrid1.DataSource =
            dsCustomers.Tables[0].DefaultView;
```


Algumas operações do DataAdapter

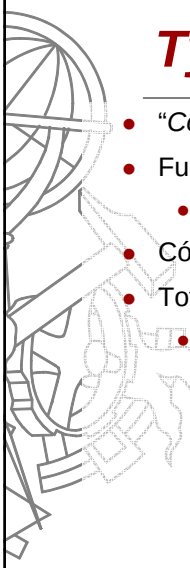
- Deve-se usar o método **Merge** para juntar dois DataSets – um original e um que contenha apenas as mudanças ao original
- O método de **Update** de um **DataAdapter** chama o comando apropriado para cada linha alterada (INSERT, UPDATE, DELETE) numa DataTable específica
- O método **AcceptChanges** de um **DataSet** efectua o **Commit** a todas as alterações efectuadas a um **DataSet** específico desde o seu último carregamento ou desde que foi chamado este método pela última vez



Typed DataSets

Visual Studio .NET pode gerar "*strongly typed* DataSets"

- Classes customisadas derivadas de **DataSet**
  - Proporciona toda a funcionalidade dos DataSet
- Contem classes, métodos, eventos e propriedades especificas relacionadas com os dados em questão
- Por exemplo, um "typed DataSet" pode conter:
  - Uma classe chamada "Customers" deriveda de DataTable
  - Uma classe chamada "CustomerRow" deriveda de DataRow com propriedades "CustomerID" e "Name"



Typed DataSets - Vantagens

- "*Compile-time type checking*"
- Funcionalidades adicionais
  - Por exemplo, o método FindByCustomerID
- Código mais legível e manutenção facilitada
- Total suporte pelo IntelliSense
  - Torna o código mais legível e menos sujeito a erros

```
string s = customerDataSet.Customers[ 1 ].Name;
```

comparado com :

```
string s = (string) customerDataSet.
                Tables["Customers"].
                    Rows[ 1 ].Item[ "Name" ];
```

Typed DataSets - Criar

- Visual Studio gera *typed DataSets* a partir definições XML *schema* (XSD)
- Visual Studio oferece varias formas de gerar definições XML *schema*
  - A forma mais fácil é:
    - Seleccionar o “Add New Item” *wizard*
    - Seleccionar a *template* Data Set para adicionar um ficheiros XSD
    - Arrastar uma *table* ou uma “*stored procedure*” do “*Server Explorer*” para a área de trabalho
    - E é tudo!

DEMO 06