Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto
Instituto Politécnico do Porto

ADO.NET

Nuno Ferreira

—♦—

Outubro de 2004

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto (ISEP/IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 PORTO
Portugal
Tel. +351 228 340 500
Fax +351 228 325 219

Criado em Setembro, 2004
Última modificação em 08 Outubro, 2004 **(v 0.3)**
Email: nacf@dei.isep.ipp.pt
URL: http://www.dei.isep.ipp.pt/~nacf

Índice

Índice de Figuras

1 Introdução

Este documento pretende servir como guia introdutório ao desenvolvimento de aplicações em Visual Studio .NET com recurso a ADO.NET.

2 O ADO.NET

2.1 Introdução

O ADO.NET fornece acesso consistente a fontes de dados, como por exemplo o SQL Server, assim como a outras fontes acessíveis via OLE DB, XML ou ODBC. As aplicações podem utilizar o ADO.NET para estabelecer ligações a essas fontes de dados de modo a recuperar, manipular e actualizar os dados.

Os resultados obtidos através da execução de comando através do ADO.NET podem ser processados directamente ou colocados num objecto ADO.NET DataSet. Este tipo de objecto permite efectuar um conjunto de operações tais como combinar dados de múltiplas fontes, estabelecer relações entre tabelas, manipular como um conjunto a estrutura da informação, etc..

As classes para trabalhar com o ADO.NET estão no `System.Data.xxxx`, em que `xxxx` refere-se à especialização do fornecedor de acesso aos dados.

Deste modo, podemos dizer que o ADO.NET é um conjunto de classes para trabalhar com dados.

2.2 Principais características:

- Um sucessor do ADO mais flexível
- Um sistema desenhado para ambientes desconectados
- Um modelo de programação com suporte avançado para XML
- Um conjunto de classes, interfaces, estruturas e enumerações que gerem o acesso a dados dentro do *framework*

2.3 Desenho de aplicações centradas em dados:

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, …

2.4 Fornecedores de Dados para ADO.NET

Um fornecedor de dados para ADO.NET é um conjunto de classes que pode ser usado para efectuar ligações a bases de dados, manipular e actualizar os dados. Seguem-se alguns exemplos de fornecedores:

SQL Server .NET Data Provider

OLE DB .NET Data Provider

ODBC .NET Data Provider

Outros (DB2/400, MySQL, …)

2.4.1 Classes típicas dos fornecedores de dados

- 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

2.5 Trabalhando com 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;

2.5.1 Utilização das classes ADO.NET num cenário conectado

Num cenário conectado, os recursos são mantidos no servidor até a ligação ser fechada (exemplo usando SQL Server Provider).

1. **Abrir ligação**

```
SqlConnection conn = new SqlConnection("SERVER=SQLSERVER; INTEGRATED
SECURITY = TRUE; INITIAL CATALOG=ISEP");
conn.Open();
```

2. **Executar comando**

```
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = conn;
    cmd.CommandText = "SELECT cod, descricao FROM detalhes WHERE
zona=42";
    reader = cmd.ExecuteReader();
```

3. **Processar linhas no reader**

```
    while(reader.Read())
    {
          int cod = (int)reader[0];
          cmbDescricao.Items.Add(reader[1].ToString());

    }
```

4. **Fechar reader**

```
if(!reader.IsClosed)
      reader.Close();
```

5. **Fechar ligação**

```
conn.Close();
```

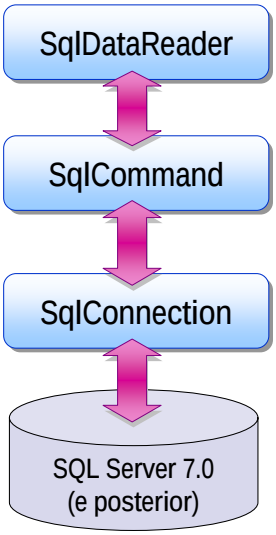

Figura 1- Cenário Conectado

2.6 Trabalhando com 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;

2.6.1 Utilização das classes ADO.NET num cenário desconectado

Num cenário desconectado, os recursos não são mantidos no servidor durante o processamento dos dados (exemplo usando SQL Server Provider).

1. **Abrir a ligação**

```
SqlConnection conn = new SqlConnection("SERVER=SQLSERVER; INTEGRATED
SECURITY = TRUE; INITIAL CATALOG=ISEP");
conn.Open();
```

2. **Preencher o DataSet**

```
System.Data.DataSet ds = new System.Data.DataSet();
System.Data.Sql.SqlDataAdapter da =
                    new System.Data.Sql.SqlDataAdapter();
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "SELECT * FROM [DETALHES]";
cmd.Connection = conn;
da.SelectCommand = cmd;
da.Fill(ds);
```

3. **Fechar a ligação**

```
conn.Close();
```

4. **Processar o DataSet**

```
foreach(DataRow r in ds.Tables[0].Rows)
   r[“preco”] = r[preco] * 1.05;
```

5. **Abrir a ligação**

```
conn.Open();
```

6. **Actualizar a fonte de dados**

```
System.Data.Sql.SqlDataAdapter da =
            new System.Data.Sql.SqlDataAdapter(
                            "SELECT * FROM [DETALHES]", conn);
System.Data.Sql.SqlCommandBuilder cb =
                new System.Data.Sql.SqlCommandBuilder(da);
da.Update(ds);
```

7. **Fechar a ligação**

```
conn.Close();
```

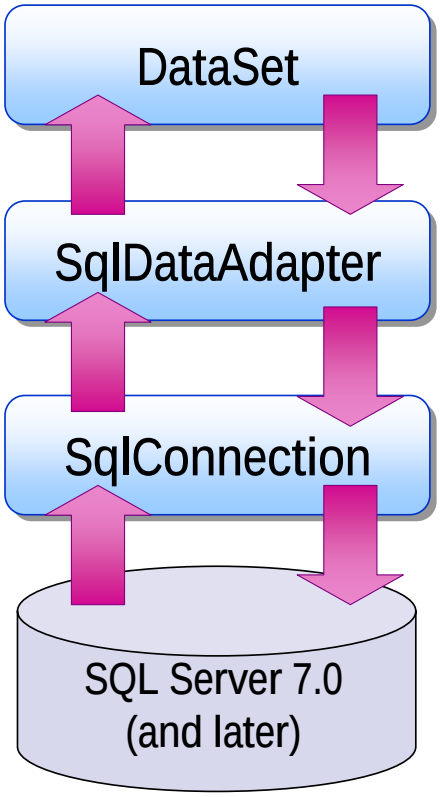

Figura 2 - Cenário Desconectado

2.7 Namespaces necessários

- `System.Data` – para as classes base do ADO.net (ex, DataSet).
- `System.Data.SqlClient` – para as classes correspondentes ao *provider* para SQL Server;
- `System.Data.OleDb` – para as classes correspondentes ao *provider* para OLE DB;
- `System.Data.SqlTypes` – para as classes correspondentes aos tipos de dados nativos do SQL Server~

- `System.Data.Common` – para as estruturas de dados, classes e interfaces comuns a todos os *providers* (ex, DataSet)
- `System.Xml` – para as classes de manipulação de XML via DataSet;

2.8 Evolução do ADO para ADO .NET

No ADO basicamente só trabalhávamos com 3 tipos de objectos: Connection, Command e Recordset.

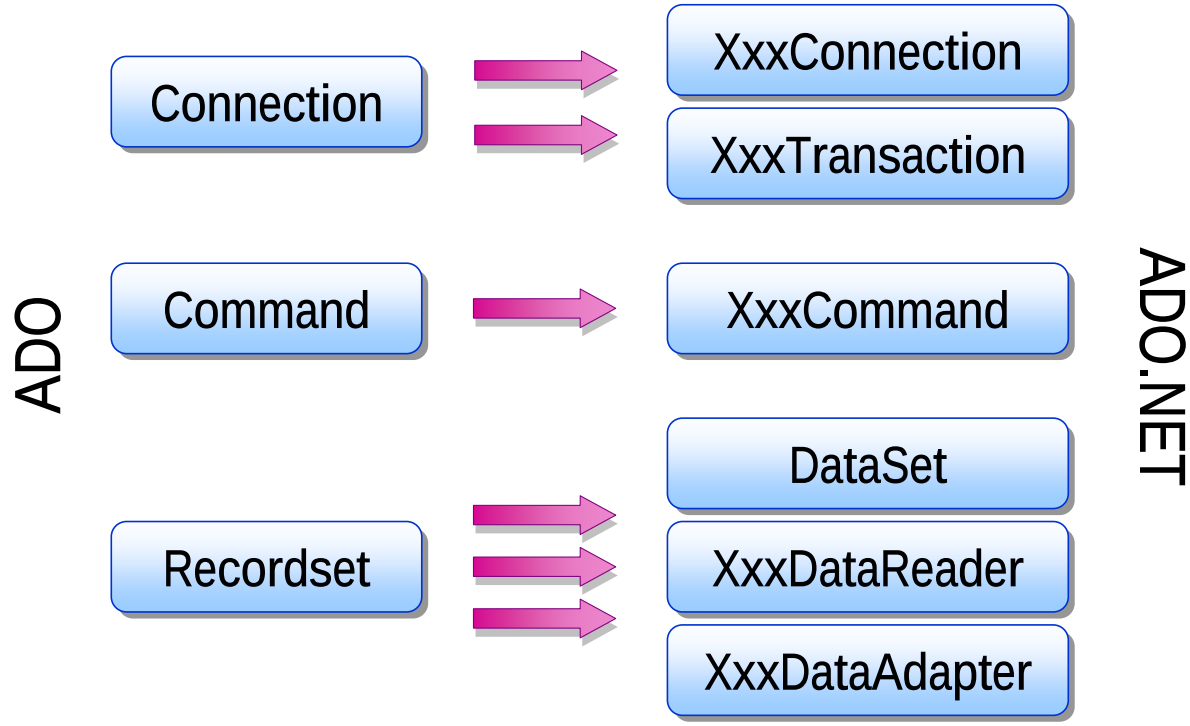

Figura 3 - Evolução do ADO para ADO.NET

No ADO.NET esses objectos foram especificados e expandidos para suportarem funcionalidades específicas dos fornecedores de acesso. No caso da figura 1, o Xxx pode ser alterado para Odbc, Sql ou outro fornecedor. Também na parte de armazenamento e manipulação dos dados houve grandes alterações, uma vez que há mais objectos disponibilizados e com mais capacidades, como o caso do DataSet, do DataReader e do DataAdapter.

2.9 Gestão de ligações com o ADO.NET

- 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

2.9.1 Como definir o comando de ligação a um fornecedor de dados

- Criar um ficheiro com extensão “UDL”
- Abrir o ficheiro e escolher o fornecedor

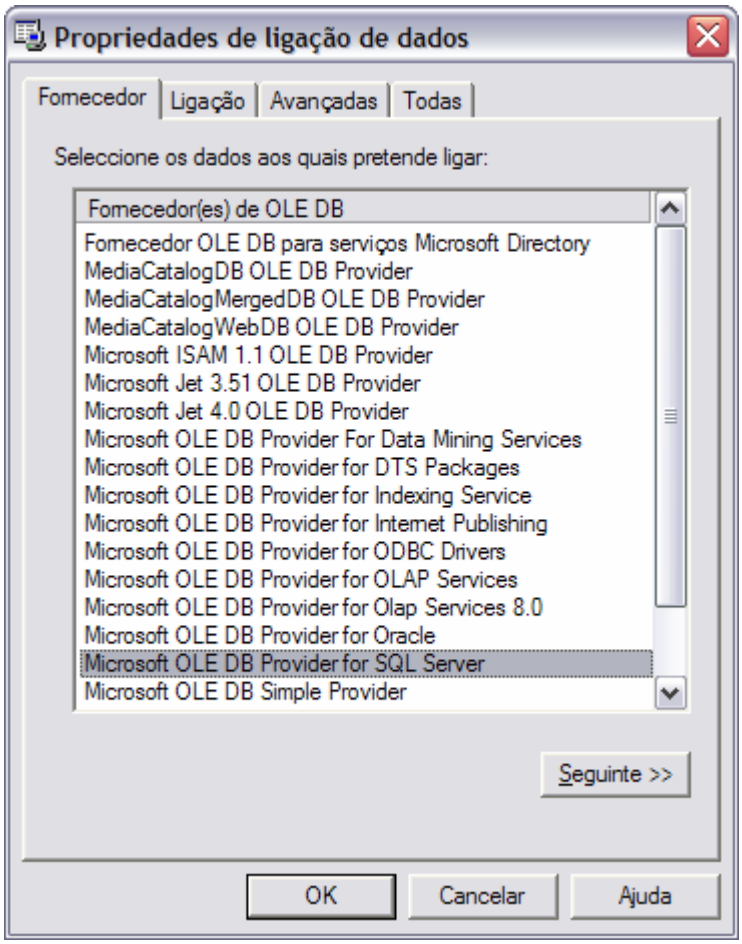

Figura 4 – Escolha Fornecedor OleDb para SQL Server

- Com base no fornecedor escolhido, preencher as opções de ligação e testar a ligação

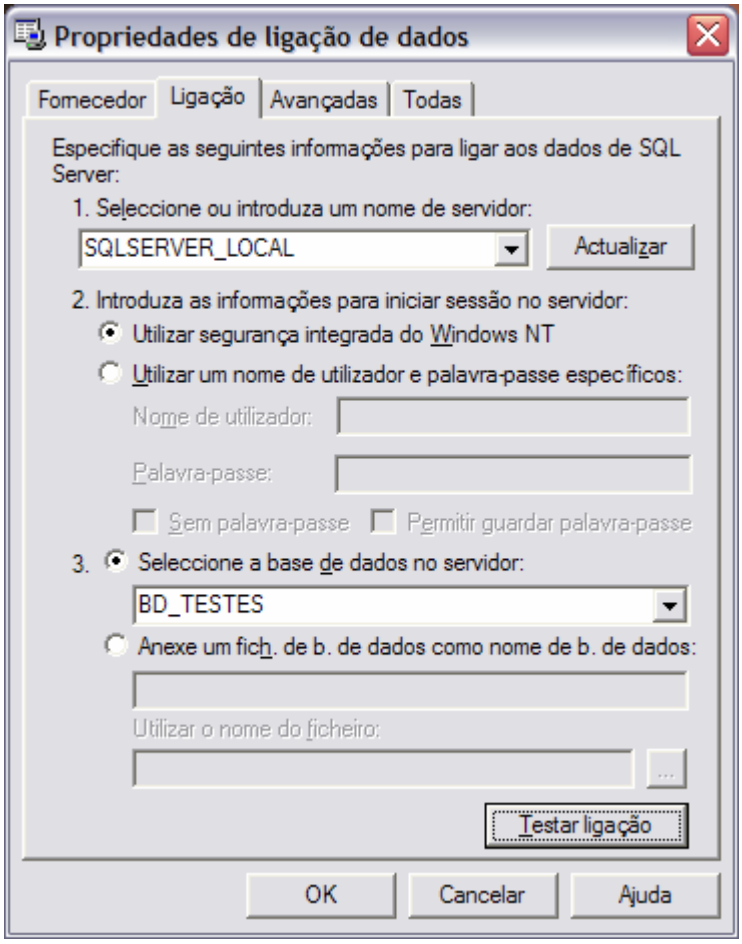

Figura 5 – Opções de ligação

- Fechar a janela e abrir o ficheiro em qualquer editor de texto. Copiar o comando.

```
Provider=SQLOLEDB.1; Integrated Security=SSPI; Persist Security Info=False;
Initial Catalog=BD_TESTES; Data Source=SQLSERVER_LOCAL
```

NOTA: *consultar exemplos de connection strings poara diferentes fontes de dados usando diferentes providers em http://www.able-consulting.com/ADO_Conn.htm*

3 Exemplos de utilização

Com estes exemplos de utilização pretende-se demonstrar algumas das aplicações práticas do ADO.NET e ao mesmo tempo explicar como se executam as tarefas mais rotineiras.

3.1 Executar comandos que retornem um só registo

```
// Definir uma ligação a um fornecedor do tipo OLEDB para Access
OleDbConnection conn = new
OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
C:\DSN\Teste.mdb;Persist Security Info=False");

// Abrir a ligação
conn.Open();

// Definir um novo comando do tipo OLEDB
OleDbCommand cmd = new OleDbCommand();
// Colocar o texto do comando
cmd.CommandText = "SELECT NOME FROM PESSOA WHERE NUMERO = ?";
// Indicar ao comando qual é a ligação que vai usar
cmd.Connection = conn;

// Definir um parametro do tipo inteiro para conter o "Número"
OleDbParameter parm = cmd.Parameters.Add(new OleDbParameter("@Numero",
OleDbType.Integer));
// Colocar o valor do parametro "Número". Quero saber o nome do cliente
cujo código é 1...
cmd.Parameters["@Numero"].Value=1;

// Executar o comando que só irá retornar um valor
// Converter o resultado numa string
// Colocar o valor de retorno na respectiva caixa de texto
txtNome.Text = cmd.ExecuteScalar().ToString();

// Fechar a ligação
conn.Close();
```

A ver: *Experimente usar o ExecuteScalar para retornar o resultado de um COUNT, MAX, MIN ou outra função semelhante.*

3.2 Executar comandos que não retornem registos (inserir, actualizar ou remover registos)

```
// Definir uma ligação a um fornecedor do tipo OLEDB para Access
OleDbConnection conn = new
OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
C:\DSN\Teste.mdb;Persist Security Info=False");

// Abrir a ligação
conn.Open();

// Definir um novo comando do tipo OLEDB
OleDbCommand cmd = new OleDbCommand();
// Colocar o texto do comando
cmd.CommandText = "INSERT INTO TRABALHOS VALUES(?, ?)";
// Indicar ao comando qual é a ligação que vai usar
cmd.Connection = conn;

// Definir os parametros para inserir os valores
OleDbParameter parmNumero = cmd.Parameters.Add(new
OleDbParameter("@Numero", OleDbType.Integer));
```

```
OleDbParameter parmNome = cmd.Parameters.Add(new OleDbParameter("@Nome",
OleDbType.Char));

// Inserir os valores
for(int i=0; i<cmbNomes.Items.Count; i++)
{
      parmNumero.Value = i+1;
      parmNome.Value = cmbNomes.Items[i];

      // Executar o comando para inserir os valores
      cmd.ExecuteNonQuery();
}

// Fechar a ligação
conn.Close();
```

A ver: *O exemplo apresentado apenas permite efectuar a inserção de registos. Experimente as capacidades de remoção e actualização, mostrando ao utilizador o número de registos que foram afectados pela operação.*

3.3 Executar comandos que retornem registos para preenchimento de informação

```
// Definir uma ligação a um fornecedor do tipo OLEDB para Access
OleDbConnection conn = new
OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
C:\DSN\Teste.mdb;Persist Security Info=False");

// Abrir a ligação
conn.Open();

// Definir um novo comando do tipo OLEDB
OleDbCommand cmd = new OleDbCommand();
// Colocar o texto do comando
cmd.CommandText = "SELECT NOME FROM PESSOA";
// Indicar ao comando qual é a ligação que vai usar
cmd.Connection = conn;

// Definir um DataReader para ler os dados
// DataReader = forward only, read only. Muito rápido.
// Executar o comando e associá-lo ao reader
OleDbDataReader reader = cmd.ExecuteReader();

// Percorrer o reader e colocar os valores
while(reader.Read())
      cmbNomes.Items.Add(reader[0].ToString());

// Se o reader não estiver fechado, fechar...
if( !reader.IsClosed )
      reader.Close();

// Fechar a ligação
conn.Close();
```

3.4 Utilização de DataSet

Com os exemplos seguintes pretende-se demonstrar algumas das potencialidades da utilização de DataSet, através da sua criação dinâmica ou através de código.

3.4.1 Criar um DataSet programaticamente

```
// Definir um DataSet chamado "AULAS"
DataSet dsAulas = new DataSet("AULAS");

// Definir as DataTable
DataTable dtAlunos;
DataTable dtInscricoes;

// Dizer que as tabelas pertencem ao DataSet
dtAlunos = dsAulas.Tables.Add("ALUNOS");
dtInscricoes = dsAulas.Tables.Add("INSCRICOES");

// Definir a estrutura das tabelas
dtAlunos.Columns.Add("NUMERO", typeof(int));
dtAlunos.Columns.Add("NOME", typeof(string));

dtInscricoes.Columns.Add("NUMERO_ALUNO", typeof(int));
dtInscricoes.Columns.Add("NUMERO_DISCIPLINA", typeof(int));

// Definir as chaves primárias das tabelas
dtAlunos.Constraints.Add("PK_ALUNOS", dtAlunos.Columns["NUMERO"], true);
dtInscricoes.Constraints.Add("PK_INSCRICOES", new
DataColumn[]{dtInscricoes.Columns["NUMERO_ALUNO"],
dtInscricoes.Columns["NUMERO_DISCIPLINA"]}, true);

// Definir as relações entre as tabelas
dsAulas.Relations.Add("R_ALUNO_DISCIPLINAS",
      dtAlunos.Columns["NUMERO"], dtInscricoes.Columns["NUMERO_ALUNO"]);

// Mostrar o DataSet numa grelha
dgAulas.DataSource = dsAulas;

// Mostrar uma tabela específica do DataSet
dgAulas.DataMember = "ALUNOS";
```

Nota: As *DataTable* têm um método de *Select* que permite filtrar as *DataRow* de modo a cumprir com restrições de ordenação e depois de estado. Tem 3 parâmetros opcionais:

- Expressões de filtragem, como por exemplo, "City=‘Porto'"
- Ordenar, por exemplo, "City ASC"
- *DataViewRowState*, por exemplo, *Deleted*

3.4.2 *XML* e *DataSet*

Podemos converter um *DataSet* num ficheiro *XML* e do mesmo modo, um *XML* num *DataSet:*

- Método ReadXML

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

- Método WriteXML

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

```
<?xml version="1.0" standalone="yes" ?>
- <AULAS>
- <xs:schema id="AULAS" xmlns=""
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
  microsoft-com:xml-msdata">
- <xs:element name="AULAS" msdata:IsDataSet="true" msdata:Locale="pt-PT">
- <xs:complexType>
- <xs:choice maxOccurs="unbounded">
- <xs:element name="ALUNOS">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="NUMERO" type="xs:int" />
  <xs:element name="NOME" type="xs:string" minOccurs="0" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
+ <xs:element name="INSCRICOES">
    </xs:choice>
    </xs:complexType>
- <xs:unique name="PK_ALUNOS" msdata:PrimaryKey="true">
  <xs:selector xpath=".//ALUNOS" />
  <xs:field xpath="NUMERO" />
    </xs:unique>
- <xs:unique name="PK_INSCRICOES" msdata:PrimaryKey="true">
  <xs:selector xpath=".//INSCRICOES" />
  <xs:field xpath="NUMERO_ALUNO" />
  <xs:field xpath="NUMERO_DISCIPLINA" />
    </xs:unique>
- <xs:keyref name="R_ALUNO_DISCIPLINAS" refer="PK_ALUNOS">
  <xs:selector xpath=".//INSCRICOES" />
  <xs:field xpath="NUMERO_ALUNO" />
    </xs:keyref>
    </xs:element>
    </xs:schema>
+ <ALUNOS>
+ <ALUNOS>
+ <ALUNOS>
+ <INSCRICOES>
```

```
+ <INSCRICOES>
  </AULAS>
```

3.5 Criar um DataView

```
// Inserir alguns registos na tabela "ALUNOS"
dtAlunos.Rows.Add(new Object[]{1, "Rui"});
dtAlunos.Rows.Add(new Object[]{2, "Ana"});
dtAlunos.Rows.Add(new Object[]{3, "Margarida"});

// Definir a DataView a indicar que é uma visão da tabela "ALUNOS"
DataView dvAlunos = new DataView(dtAlunos);

// Ordenar por nome
dvAlunos.Sort = "NOME";

// Associar esta DataView à grelha, para visualizar os resultados
dgAulas.DataSource = dvAlunos;
```

A ver: *O DataView tem outras características, como por exemplo, a possibilidade de filtrar os dados, por conteúdo e por estado. Para testar esta característica, crie uma segunda grelha que mostre os registos que foram apagados da tabela…*

Nota:

- O método *Remove* da classe *DataRowCollection* apaga completamente o registo da colecção;
- O método *Delete* da classe *DataRow* marca o registo como apagado. Este fica escondido, mas acessível, se necessário.
- 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

3.6 Percorrer registos de uma DataTable

Para percorrer registos de uma *DataTable* necessitamos de associar um objecto chamado *CurrencyManager* à tabela. Este objecto permite manter a posição do cursor dentro da tabela. O primeiro registo tem a posição 0. Cada *DataTable* ou *DataView* apenas pode ter um só *CurrencyManager*.

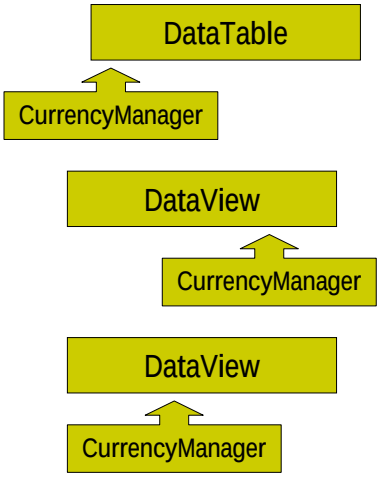

Figura 6 – CurrencyManager

```
CurrencyManager cm;
cm = (CurrencyManager)this.BindingContext[dsGestaoBanco1, "Contas"];
cm.Position += 5;
```

4 Exercício

Desenvolva uma aplicação utilizando a linguagem .net da sua preferência e ADO.NET que lhe permita efectuar a manutenção de uma base de dados simples com os seus registos bancários. Utilize o seguinte esquema de base de dados:

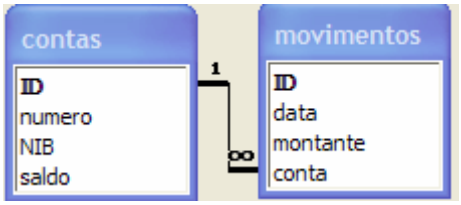

Figura 7 – Base de Dados para trabalho

Esta aplicação deverá carregar numa *combo box* todos os números de contas bancárias e, após o utilizador escolher um determinado número, mostrar numa grelha todos os movimentos dessa conta. Deverá também permitir ter sempre actualizado o saldo.

5 Informação Adicional

Laboratório .net do ISEP/IPP
http://www.dei.isep.ipp.pt/labdotnet/
MSDN Library
http://msdn.microsoft.com/library
.net framework center

http://msdn.microsoft.com/netframework/
C#
http://msdn.microsoft.com/vcsharp/
Open CLI
http://sourceforge.net/projects/ocl
Mono (.net @ Unix)
http://www.go-mono.com/
ECMA
http://www.ecma-international.org/
Introduction to C# @ ECMA
http://www.ecma-international.org/activities/Languages/Introduction%20to%20Csharp.pdf
Common Language Infrastructure @ ECMA
http://www.ecma-international.org/activities/Languages/ECMA%20CLI%20Presentation.pdf
ASP.net
http://www.asp.net
Winforms
http://www.windowsforms.net/
Using ADO.net
http://msdn.microsoft.com/netframework/using/understanding/data/default.aspx?pull=/library/en-us/dndotnet/html/usingadonet.asp
Introduction to .Net Remoting
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/introremoting.asp?frame=true