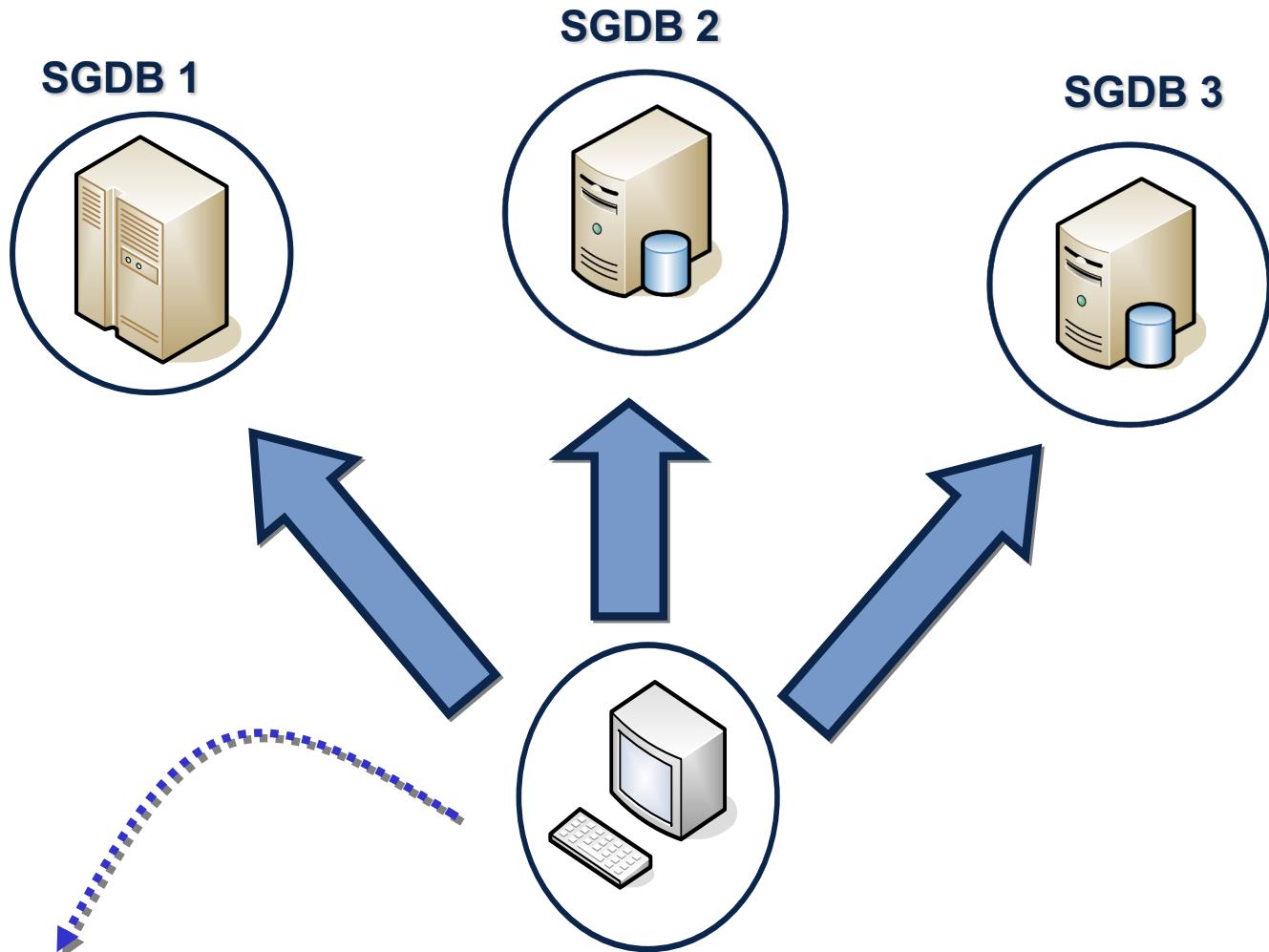

ADO.NET

Agenda

- Evolução tecnológica no acesso a dados
- Vista geral sobre o ADO.NET
- *Connected Classes*
- *Disconnect Classes*
- Modelo Transaccional

Evolução da tecnologia de acesso a dados



Objectivo: Desenvolver código de acesso a dados independente do SGDB

Cenário: Década de 80

Evolução da tecnologia de acesso a dados

1980

1985

1990

2000

- Não existia uma infra-estrutura de acesso a dados independente do SGBD
- O acesso a diferentes fontes de dados necessitavam algum código e muito esforço
- As ferramentas escolhidas eram, principalmente, SQL embutido e pré-processadores



- Tecnologia proprietária
- Flexibilidade limitada

Evolução da tecnologia de acesso a dados

1980

1990

1992

2000

- Como resposta a estes problemas, foi fundado em 1989 um consórcio, *SQL Access Group* (SAG), com o objectivo de fomentar a interoperabilidade e potabilidade entre diferentes SGBD
- Em 1992 surge a primeira definição do ODBC (*Open DataBase Connectivity*) que viria a revolucionar o acesso a dados no mundo PC
- Permitiu que os PC's se tornassem “os clientes” para as aplicações empresariais
- O ponto de viragem



Evolução da tecnologia de acesso a dados

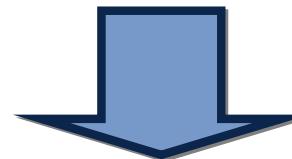
1980

1990

1992

2000

- O ODBC teve várias versões, a última em 1996 – ODBC 3.0
- Existem várias implementações da especificação
 - *Microsoft ODBC*
 - *iODBC*
 - *UnixODBC*
- O ODBC está demasiado ligado à linguagem SQL, o que levantou problemas no acesso a fontes de dados não relacionais (e.g. Multidimensionais)



Evolução da tecnologia de acesso a dados

1980

- Mas paralelamente ao envolvimento com o *ODBC*, a Microsoft foi lançando API's de acesso a dados, com o objectivo dar suporte a novas necessidades – *DAO* (*DataAccess Objects*) e *RDO* (*Remote Data Objects*)

1990

- Surge em 1996 o *JDBC*, como forma standard de aceder a dados em Java

1992

- O desenvolvimento de aplicações baseada em componentes começava a emergir

1996

A Microsoft criou uma nova tecnologia de acesso a dados, suportada pela sua tecnologia de componentes (*Common Object Model - COM*)

2000

Evolução da tecnologia de acesso a dados

1980

1990

1996

2000

- Em 1996 surgiu o OLE-DB (*Object Linking and Embedding for DataBase*)
- OLE-DB é uma especificação para acesso a dados que implementa o conceito produtor/consumidor – semelhante ao ODBC
- As vantagens eram obvias
 - Independência do SQL e do modelo relacional
 - Utilização da tecnologia COM
- Além do mais, possuía a utilização do ODBC



Evolução da tecnologia de acesso a dados

1980

- No entanto o OLE tinha um modelo de programação muito próximo do COM
- Desenvolver código utilizando OLE-DB não estava ao alcance de todos os programadores, visto ser necessário o domínio da linguagem C++
- Nasce então uma API de abstracção que se coloca por cima do OLE-DB mas que apresenta um modelo de programação mais simples e intuitivo
- Surge no final de 1995 *Active Data Objects* (ADO)

1990

1996

2000

Evolução da tecnologia de acesso a dados

1980

- O ADO possibilitou o acesso a dados a um novo mundo de aplicações
 - Devido à simplicidade do modelo de programação, à sua independência do C++, foi adoptada como “a tecnologia de acesso a dados” no mundo Windows
- 
- No entanto a evolução da tecnologia e as novas necessidades, vieram revelar algumas das fragilidades do ADO
 - Em 2001 surge o ADO.NET

1990

2000

2001

Evolução da tecnologia de acesso a dados

1980

1990

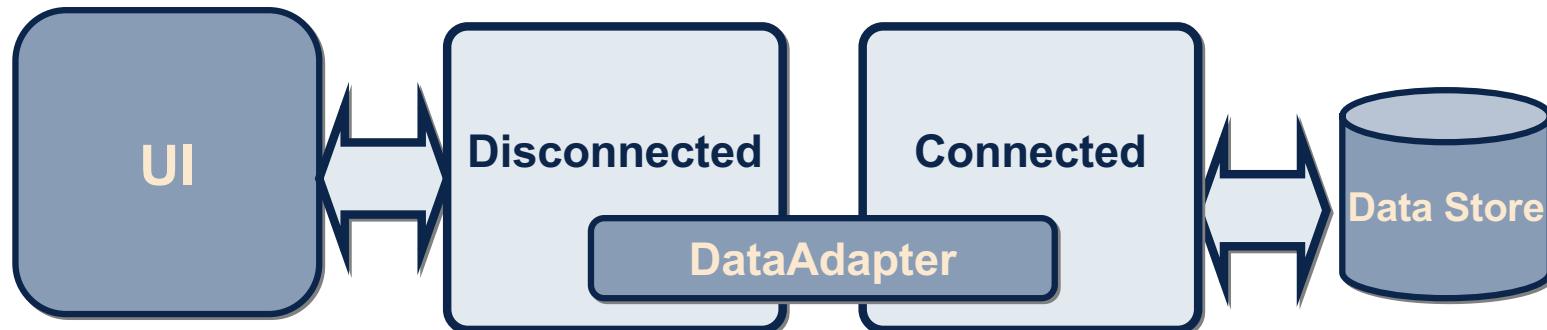
2000

2001

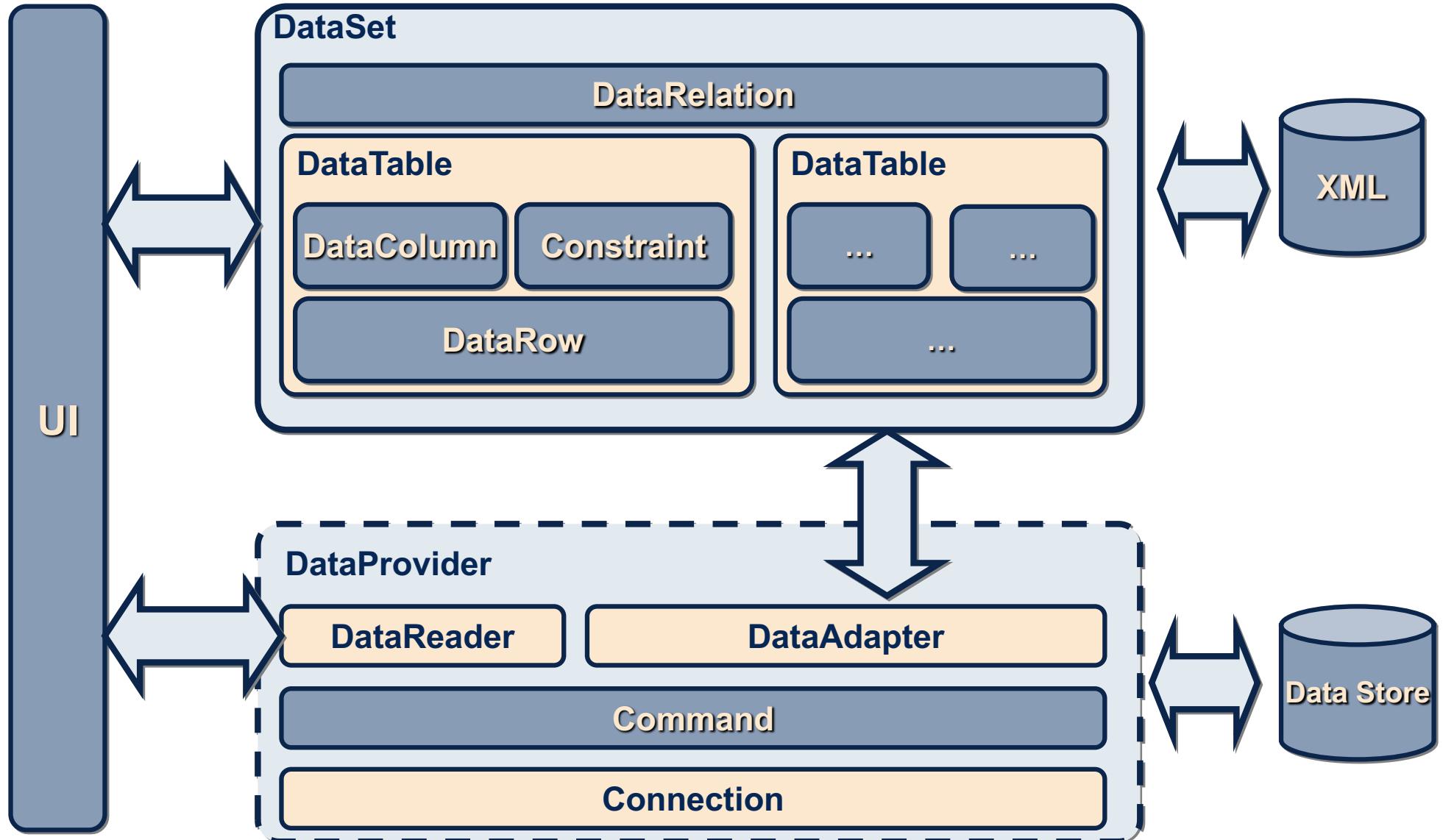
- O ADO estava intimamente ligado ao COM
- O COM sofreu uma (r)evolução, dando lugar à plataforma .NET
- E em .NET, temos um ambiente de execução controlado, que contrasta com o COM: a coexistência de tecnologias legadas não abona em favor do desempenho
- Além disso, em muitas situações, o ADO mostrava-se pouco eficiente, uma vez que o modelo de programação era inherentemente ligado

ADO.NET

- ADO.NET é parte integrante da infra-estrutura .NET
- É constituída por um conjunto de classes que exportam serviços de acesso a repositórios de dados
- A principal diferença desta tecnologia, face a outras anteriores, é o facto ser possível interagir com os dados, de forma totalmente desligada do repositório de dados
- É constituída, principalmente, por dois tipos distintos de classes
 - *Connected*
 - *Disconnected*



ADO.NET

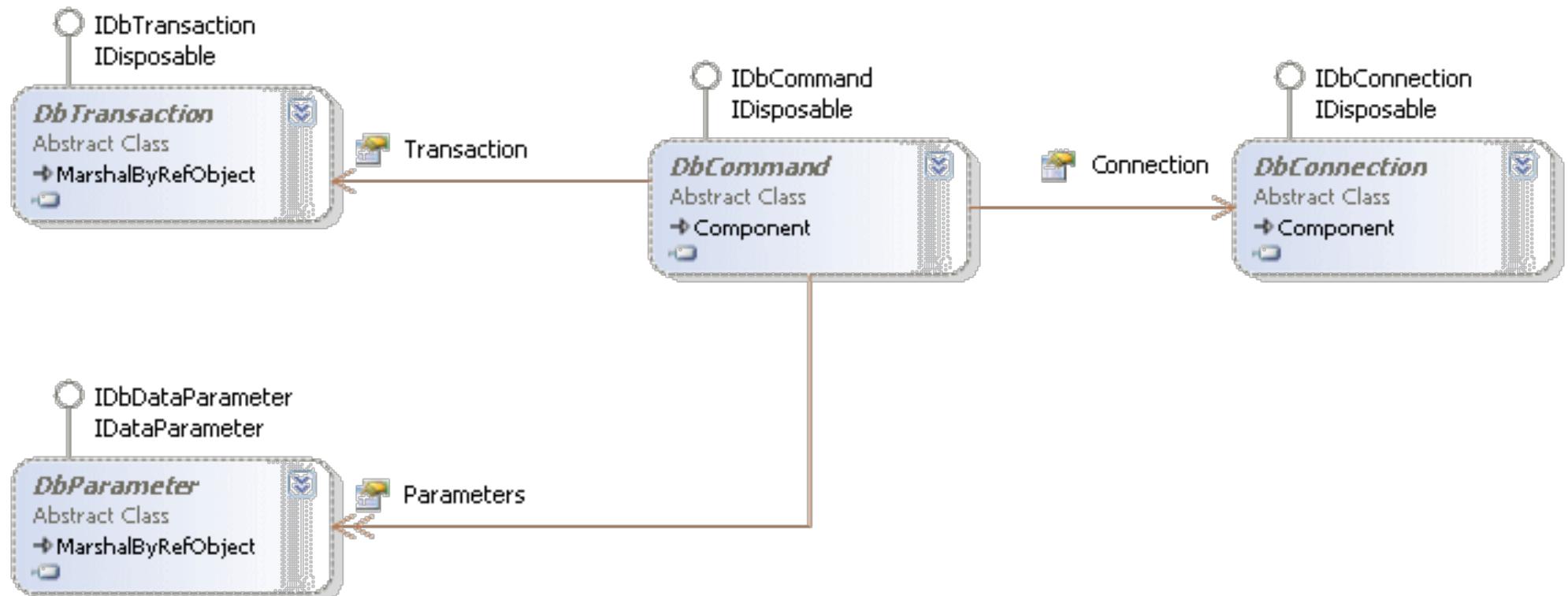


ADO.NET

- *Connected Classes*
 - *Provider-Independent APIs*
 - Controlo do *Connection pooling*
 - Integração com o *namespace* System.Transactions e promoção automática de transacções
 - Processamento Assíncrono
 - Múltiplos *Result Sets* (MARS)
- *Disconnected Classes*
 - Criação de uma *DataTable* a partir de uma *DataView*
 - Refinamentos no motor de inferência do esquema
 - Serialização do esquema em *Datasets* tipificados
 - Melhoramentos no XML/XSD

Connected Classes

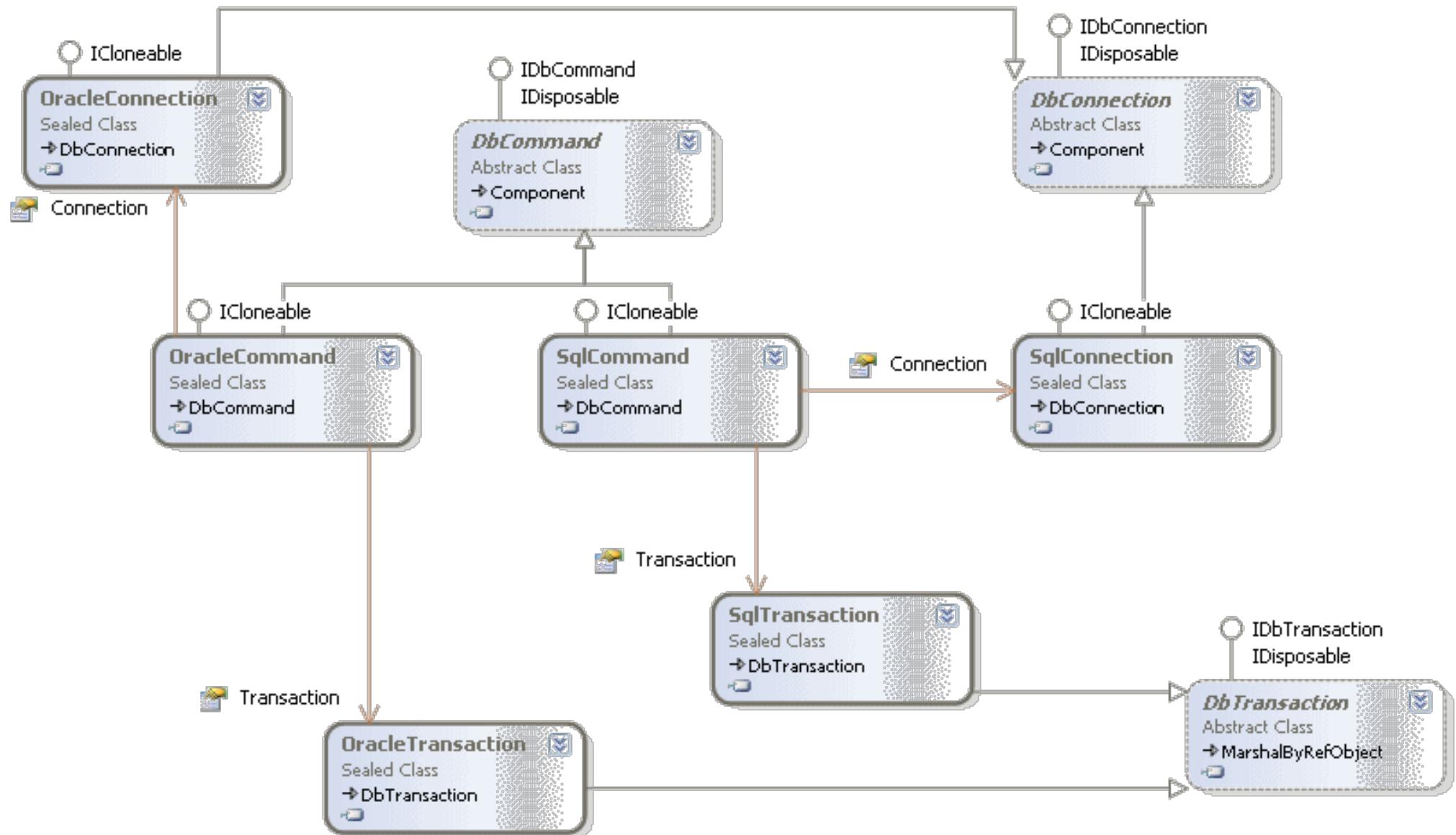
- As classes que necessitam de ligação ao repositório de dados, espelham o modelo de interacção já conhecido de outras tecnologias de acesso a dados



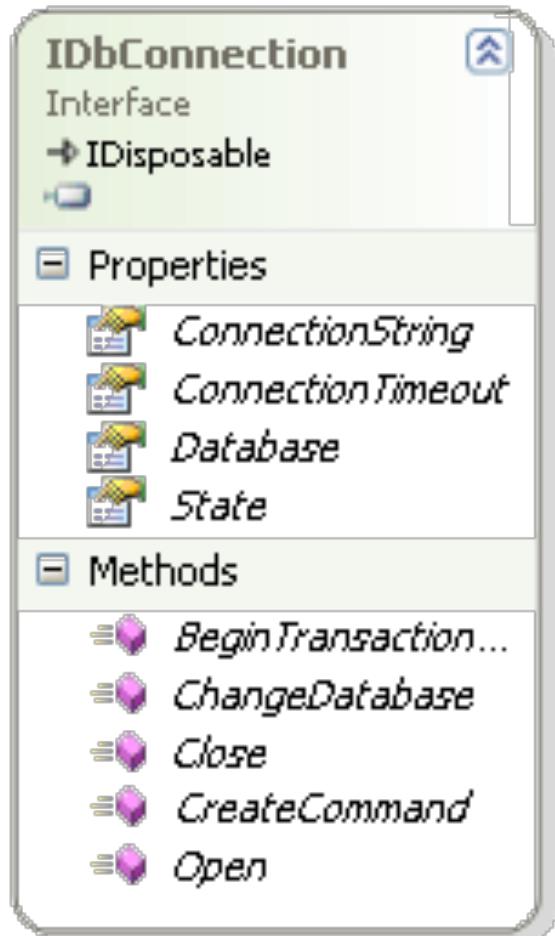
.NET Data Providers

- De notar que as classes `DbXXX` são abstractas
- Para cada tipo específico de fontes de dados, existem especializações de cada uma delas
- Como tal, uma fonte de dados específica dá origem a um *.NET Data Provider*
 - *OdbcXXX* (e.g. `OdbcConnection` e `OdbcCommand`)
 - *OleDbXXX*
 - *SqlXXX*
 - *OracleXXX*
- *Note-se que o tipo .NET está ligado a um repositório específico*

.NET Data Providers



System.Data.Common.IDbConnection

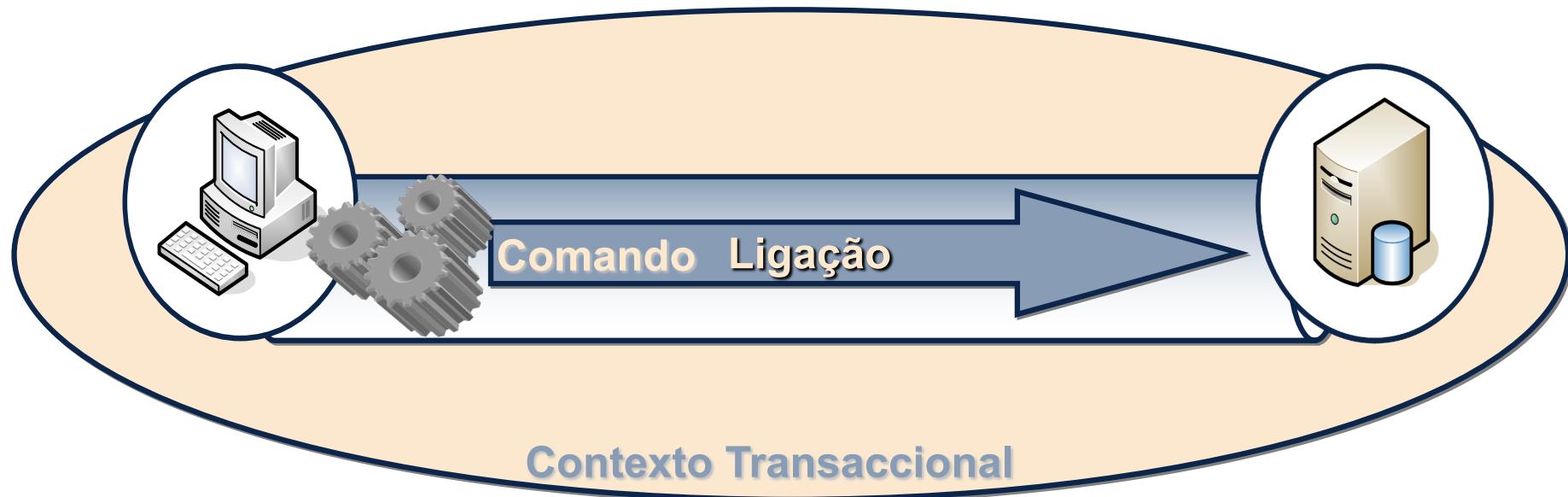


- Esta interface é implementada parcialmente pela classe `DbConnection` e define o contrato de uma ligação a um repositório de dados
- Note-se que é no âmbito de uma ligação que
 - pode ser aberta uma transacção – `BeginTransaction`
 - pode ser criado um comando – `CreateCommand`
- Isto espelha o facto de uma ligação ser o ponto central e unificador dos classes `DbXXXX`

System.Data.Common.IDbConnection

- Refira-se que no ADO.NET, o `connectionTimeout` é diferente do `commandTimeout` e, assim, não é possível configurar este último ao nível da ligação
- É também reforçada, logo na declaração da interface, a necessidade de tratar uma ligação como um recurso crítico, passível de ter impacto no desempenho do programa
- **Como tal, `IDbConnection` deriva de `IDisposable`, por forma a possibilitar a libertação de recursos pela infra-estrutura .NET**

Execução de um comando: Sequência típica de acções



1. Estabelecer uma ligação à fonte de dados
2. Especificar um comando a ser efectuado
 - Opcionalmente pode ser necessário utilizar parâmetros
3. Se necessário manipular transacções
4. Executar o comando
5. Caso o comando produza um conjunto de dados, manipular o resultado
6. Se existir um contexto transaccional activo, indicar o sucesso da acção
7. Fechar a ligação

Connected Classes: HelloWorld

```
SqlConnection con = new SqlConnection();
try
{
    con.ConnectionString= @"Data Source=.\SQLEXPRESS;
                           Initial Catalog=ADONET;Integrated Security=True";
    SqlCommand cmd = con.CreateCommand();
    cmd.CommandText = "select value from DEMO";
    con.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
        Console.WriteLine(dr["value"]);
}
catch (Exception ex)
{
    Console.WriteLine("E R R O : "+ex.Message);
}
finally
{
    if(con.State != ConnectionState.Closed)
        con.Close();
}
```

ConnectionString

- A *ConnectionString* é específica para cada *provider*
- Como tal, o conjunto de pares *<chave, valor>* é também específico (logo variável)
- Regra geral é necessário especificar sempre
 - **A fonte de dados**
 - **Informação relativa ao utilizador**
- Para um enorme conjunto de propriedades, existe um valor por omissão que, podendo ser alterado, é normalmente adequado para a maioria das aplicações

```
con.ConnectionString= @"Data Source=.\SQLEXPRESS;  
Initial Catalog=ADONET;Integrated Security=True";
```

- No exemplo acima
 - a fonte de dados foi o servidor .\SQLEXPRESS
 - o catálogo inicial é a base de dados ADONET
 - é utilizada a segurança integrada do Windows

ConnectionStrings e ficheiros de configuração

- Em muitos cenários, as *connectionStrings* são conhecidas de antemão
- Colocá-las no código tem desvantagens
 - Podem mudar quando os sistemas passam para produção
 - O código pode sofrer um *reverse engineering* e revelar as *connectionStrings* utilizadas
- Além disso, por questões de desempenho, seria desejável que a *ConnectionString* tivesse num local acessível, por forma a ser partilhada por diversos utilizadores
- Normalmente são escolhidos os ficheiros de configuração para as guardar, e.g. **web.config**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <connectionStrings>
        <add name="connectString" connectionstring="...." />
    </connectionStrings>
</configuration>
```

ConnectionStrings e ficheiros de configuração

```
ConnectionStringSettings cs =  
    ConfigurationManager.ConnectionStrings["connectionString"] ;  
SqlConnection connection = new  
    SqlConnection(cs.ConnectionString) )
```

- Através do tipo `ConfigurationManager` tem-se acesso ao ficheiro de configuração e às *ConnectionStrings*
- Para alterá-las basta modificar o ficheiro de configuração
- Subsiste ainda o problema de informação sensível ficar em claro no ficheiro de configuração
- Na versão .NET 1.x seria o programador a implementar um esquema de protecção

Fechar Ligações – Uma melhor abordagem

- Como foi mencionado, a interface `IDbConnection` estende de `IDisposable`
- O método `IDisposable.Dispose` é utilizado para tarefas de limpeza e libertação de recursos
- Assim, é preferível evocar o método `IDbConnection.Dispose` em vez de `IDbConnection.Close`
- Evocar o `IDbConnection.Close` possibilita que a ligação seja colocada no *pool*
- Evocar o `IDbConnection.Dispose` possibilita que a ligação seja colocada no *pool*, mas também que os recursos por ela mantidos sejam recolhidos pelo mecanismo de *garbage collection*

Fechar Ligações – Uma melhor abordagem

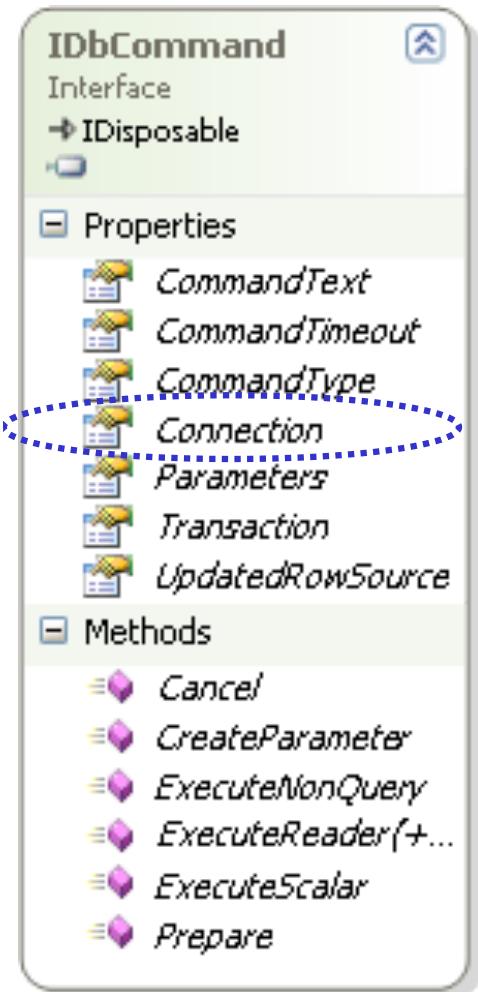
- Existem construções nas linguagens C# e VB.net que evocam implicitamente o Dispose
- O bloco `using` é um deles

```
using ( SqlConnection connection = new SqlConnection() )  
{  
    ...  
} //é evocado o método connection.Dispose();
```

HelloWorld Revisitado

```
try
{
    using(SqlConnection con = new SqlConnection())
    {
        con.ConnectionString= @"Data Source=.\SQLEXPRESS;
                                Initial Catalog=ADONET;
                                Integrated Security=True";
        using(SqlCommand cmd = con.CreateCommand())
        {
            cmd.CommandText = "select value from DEMO";
            con.Open();
            SqlDataReader dr = cmd.ExecuteReader();
            while (dr.Read())
                Console.WriteLine(dr["value"]);
        }
    }
}
catch (Exception ex)
{
    Console.WriteLine("E R R O : "+ex.Message);
}
```

System.Data.Common.IDbCommand



- À semelhança do que acontecia com `IDbConnection`, todos os **ADO.NET providers** têm uma implementação especializada de `IDbCommand`
- Esta interface define o conjunto mínimo de requisitos para executar um comando junto de um repositório de dados
- Note-se que a execução de um comando terá de ser sempre feita no âmbito de uma **ligação aberta**

Diferentes formas de criar um comando

```
using(SqlCommand cmd = con.CreateCommand())  
{  
    cmd.CommandText = "select value from DEMO";  
    con.Open();  
    SqlDataReader dr = cmd.ExecuteReader();  
    while (dr.Read())  
        Console.WriteLine(dr["value"]);  
}  
...
```

```
using(SqlCommand cmd = new SqlCommand(con))  
{  
    ...  
}
```

```
using(SqlCommand cmd =  
    new SqlCommand("select value from DEMO",con))  
{  
    ...  
}
```

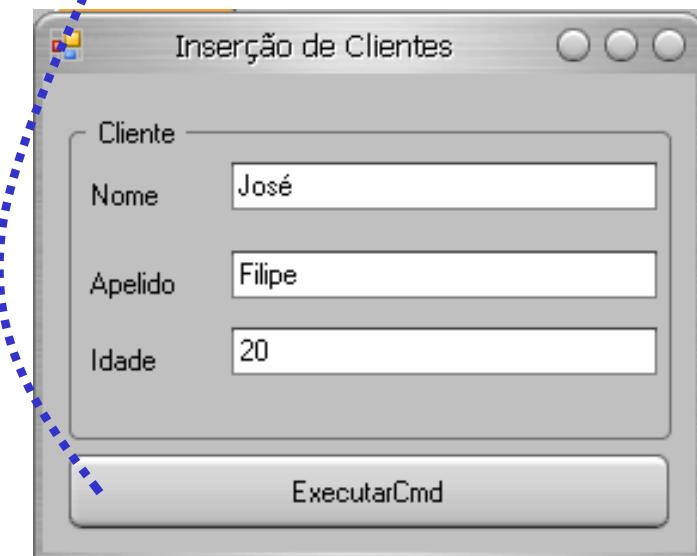
Diferentes formas de executar um comando

- O ADO.NET `DbCommand` tem vários métodos que permitem a execução de um comando
- A vantagem de utilizar o método mais adequado prende-se com uma melhor gestão de recursos

Método	Descrição
<code>ExecuteNonQuery</code>	Permite a execução de comando SQL como INSERT, DELETE, UPDATE e SET
<code>ExecuteScalar</code>	Permite executar comandos que retornem um único valor. Ex. Agregações.
<code>ExecuteReader</code>	Este método permite a execução de comandos que retomam tuplos. Por questões de performance, a execução é feita utilizando o procedimento armazenado <code>sp_executesql</code> .

DbCommand - Parâmetros

- Em muitas situações, são necessários construir comandos com base em valores recolhidos junto dos utilizadores
- Muitos dos comandos utilizam valores recolhidos dos utilizadores



```
insert into Cliente(nome,apelido,idade) values('José','Filipe',20)
```

```
insert into Cliente(nome,apelido,idade) values('Paulo','Carvalho',30)
```

Problemas

- Ataques por *SQL injection*
- Não existe uma verificação do tipo dos valores antes da execução do comando
- Impossibilidade de gerir os planos de execução e melhorar o desempenho do sistema
- ...

System.Data.Common.DbParameters

- Recorrendo a parâmetros, é possível evitar os problemas anteriores
- Muitas das verificações podem ser feitas do lado do cliente, evitando uma abertura de uma ligação para um comando que nunca será possível executar
- Para os utilizar:
 - Os parâmetros têm de ter um nome. Este **têm de começar por @**
 - É através do nome que se acede ao valor do parâmetro
 - Têm de se especificar **o tipo SQL** do parâmetro.
 - Quando necessário, deve ser feita a conversão explícita. Desta forma detecta-se incoerências antes da execução do comando

System.Data.Common.DbParameters

```
...
using(SqlCommand con = com.CreateCommand())
{
    Criar { SqlParameter nome = new SqlParameter("@nome", SqlDbType.VarChar,50);
    SqlParameter apelido= new SqlParameter("@apelido", SqlDbType.VarChar,50);
    SqlParameter idade = new SqlParameter("@idade", SqlDbType.TinyInt);

    cmd.Parameters.Add(nome);
    cmd.Parameters.Add(apelido);
    cmd.Parameters.Add(idade); } Assoc.

    nome.Value = txtNome.Text;
    apelido.Value = txtApelido.Text;
    idade.Value = Convert.ToSByte(txtIdade.Text); } Preencher

    cmd.CommandText = "insert into Cliente(nome,apelido,idade)
                      values (@nome,@apelido,@idade)";
    con.Open();
    return cmd.ExecuteNonQuery();
}
...
```

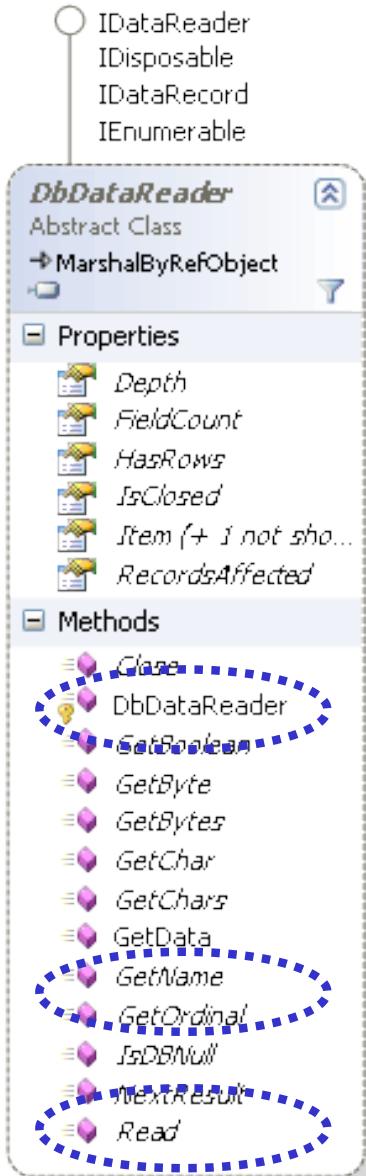
Diferentes tipos de comandos

- A classe `DbCommand` tem uma propriedade que indica a natureza do comando a ser executado – `CommandType`

Parâmetro	Descrição
<i>StoredProcedure</i>	Indica que o comando é um nome de um procedimento armazenado.
<i>TableDirect</i>	Indica que o comando é um nome de uma tabela. Apenas é suportado pelo <i>.NET provider OLE DB</i> .
<i>Text</i>	O valor por omissão. Indica que é um comando SQL.

- A correcta definição desta propriedade melhora a eficiência da execução do comando

System.Data.Common.DbDataReader



- Esta classe fornece uma implementação parcial para um cursor *server side, read-only, forward only*
- Existem implementações para os quatro principais .NET providers
- Os *DataReaders* são objectos que foram implementados tendo em conta a eficiência no acesso aos dados
- Os dados são obtidos através do método `read()`, tuplo a tuplo, para um *buffer* interno
- O acesso à informação pode ser feito utilizando um *indexer*, por índice ou por nome (menos eficiente)

System.Data.Common.DbDataReader

- No entanto o *indexer* retorna o valor de um campo como um object

```
ObjDbDataReader["UmCampo"]; // Utilizando o nome  
ObjDbDataReader[0] ; // Utilizando o índice
```

- Existem também métodos Get [Tipo], que devolvem o valor de acordo com o tipo especificado
 - Se necessitarmos de aceder a um inteiro, podemos utilizar o método `GetInt32`
- Existem métodos todos os tipos nativos .NET

```
ObjDbDataReader.GetString["MeuNome"];  
ObjDbDataReader.GetInt32[0] ;  
...
```

Agenda

- Vista geral sobre o ADO.NET
- *Connected Classes – Connection Pooling*
- *Disconnect Classes*
- Modelo Transaccional

Connection Pooling

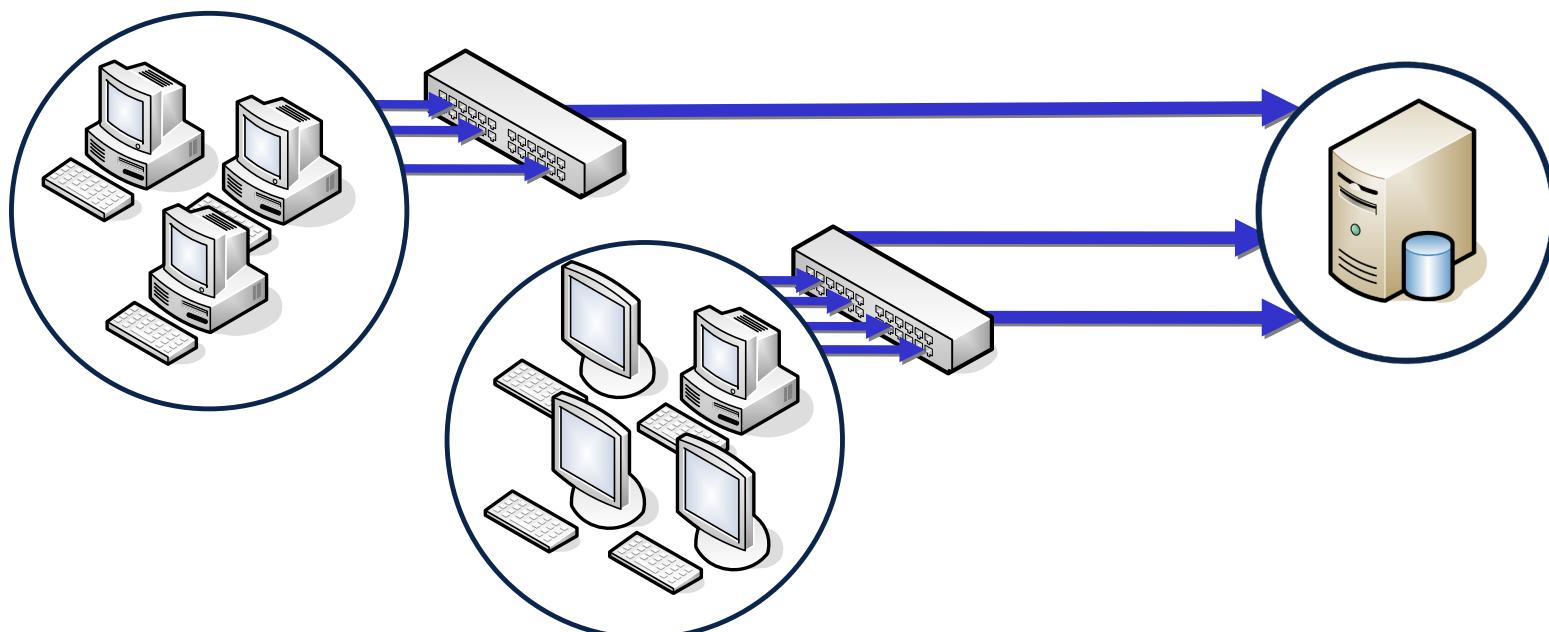
- Foque-se a nossa atenção numa aplicação que requer acesso a um repositório de dados
- Por certo que já demos por nós a esperar dois ou três segundos depois de desencadear um acesso trivial ao repositório da dados
- Este tempo é, normalmente, gasto no estabelecimento de uma ligação
- Uma solução para esta espera poderia passar por antecipar a abertura da ligação e adiar o seu fecho

Connection Pooling

- No entanto esta solução limitaria a escalabilidade da aplicação
 - Recorde-se que no limite cada utilizador poderia ter um ligação aberta para o servidor: Mas o número de ligações simultâneas é limitado
 - Se os pedidos são muitos, a infra-estrutura poderia passar a maior parte do tempo a abrir e fechar ligações
- O objectivo é que o repositório de dados passa o maior tempo possível a fornecer dados, não a gerir ligações
- Aqui entre o *connection pooling*

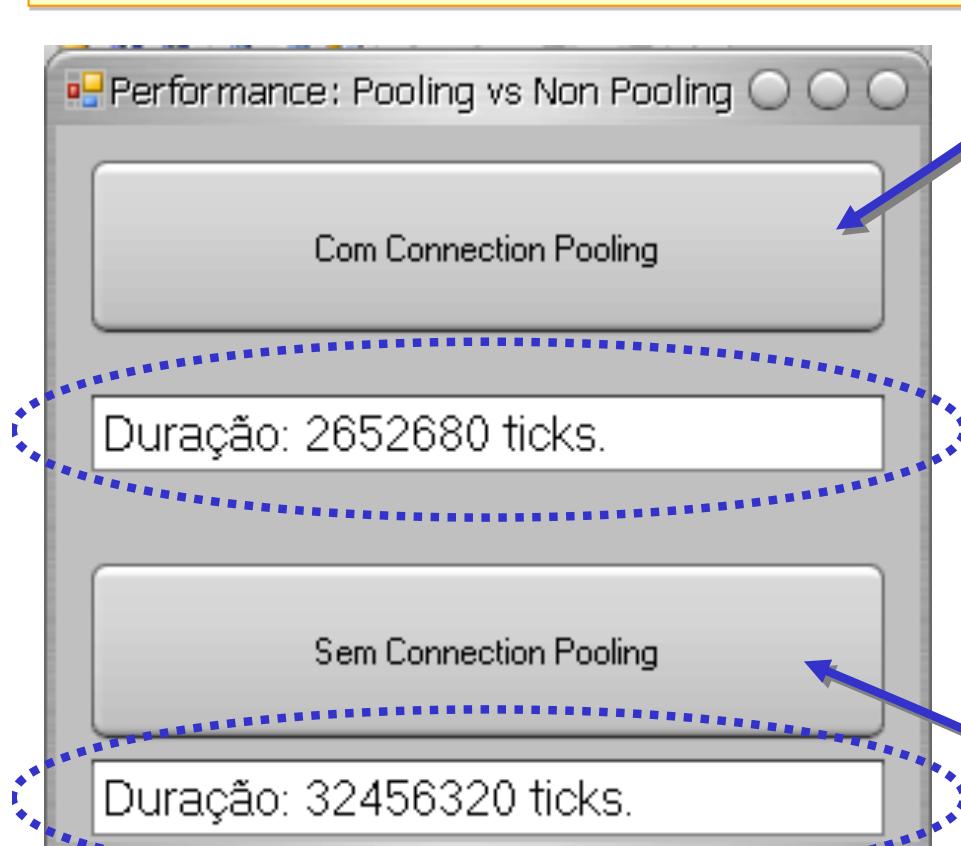
Connection Pooling

- Com *Connection pooling*, o sistema pode gerir de forma eficiente as ligações, reutilizando as que estão activas, em vez de serem criadas novas
- Para isso, existe um gestor de ligações, responsável por manter uma “lista” de ligações prontas a utilizar



Connection Pooling - Desempenho

```
... new SqlConnection("Integrated Security=TRUE;Initial Catalog=ADONET;  
Data Source=TAGUS\\SQLEXPRESS"))
```



```
startTicks = DateTime.Now.Ticks;  
  
for (int i = 1; i <= 100; i++)  
{  
    testConnection.Open();  
    testConnection.Close();  
}  
endTicks = DateTime.Now.Ticks;
```

```
... new SqlConnection("Integrated Security=TRUE;Initial Catalog=ADONET;  
Data Source=TAGUS\\SQLEXPRESS; Pooling=false))
```

Connection Pooling

- Para que o *pooling* funcione é necessário:
 - Que cada um dos caracteres da *connectionString* sejam exactamente iguais – “*Data Source=.*” É diferente de “*Data Source = .*”
 - As credenciais do utilizador também servem para diferenciar ligações “compatíveis”, mesmo que se utilize a autenticação do Windows (*Trusted_Connection=Yes*)
 - O *Process ID* têm de ser o mesmo. É impossível partilhar ligações entre processos diferentes

Connection Pooling

- Note-se que o conceito de ligação compatível é feito através da *connectionString*
- Na realidade podemos pensar no estrutura de dados subjacente ao gestor de ligações como um contentor associativo, onde
 - A chave é a *connectionString*
 - O valor associado - uma lista de ligações prontas a utilizar
- A especificação do funcionamento do *pool* é feito através de pares *<chave-valor>* enviados directamente na *connectionString*

Connection Pooling - Parâmetros

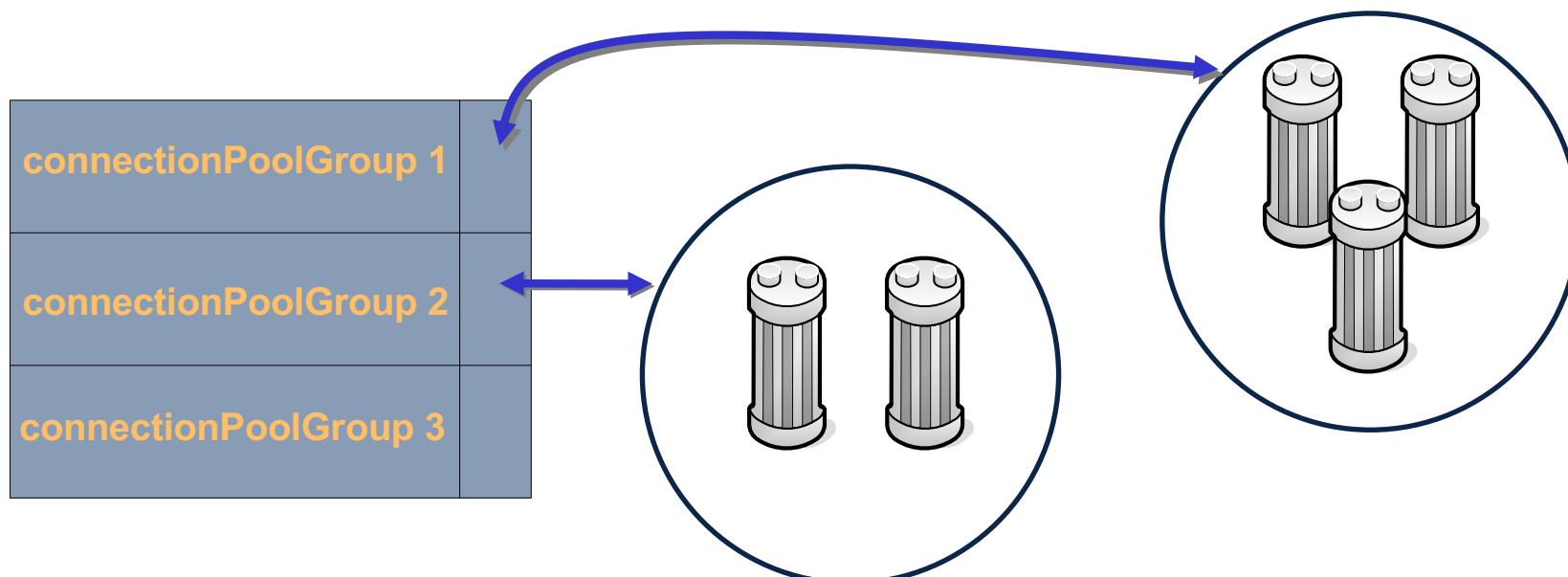
Parâmetro	Descrição
Connection Timeout	Tempo de espera por uma ligação ao repositório de dados. O valor por omissão é de 15 segundos.
Min Pool Size	O valor mínimo de ligações <i>pooled</i> . O valor por omissão é 0.
Max Pool Size	O valor máximo de ligações <i>pooled</i> . O valor por omissão é 100.
Pooling	True significa que novas ligações são retiradas do <i>pool</i> . O valor por omissão é true.
Connection Reset	Indica se é feito <i>reset</i> à ligação ao servidor de dados, sempre que uma ligação sair do pool. O valor por omissão é true.
Load Balancing timeout (former Connection Lifetime)	O tempo máximo que uma ligação do <i>pool</i> deve existir. O valor por omissão é 0. (Sem limite)
Enlist	Indica se a ligação é associada ao contexto transaccional corrente. O valor por omissão é true.

Connection Pooling

- O *connection pooling* está localizado no lado do cliente
- Todo o processo é gerido na máquina onde se inicia um objecto `DbConnection`
- As ligações *pooled* são por *ADO.NET Provider*
- Por cada *connectionString* diferente é criado um *connectionPoolGroup*
- Da primeira vez que uma ligação é criada (o objecto), é iniciado o *connectionPoolGroup*

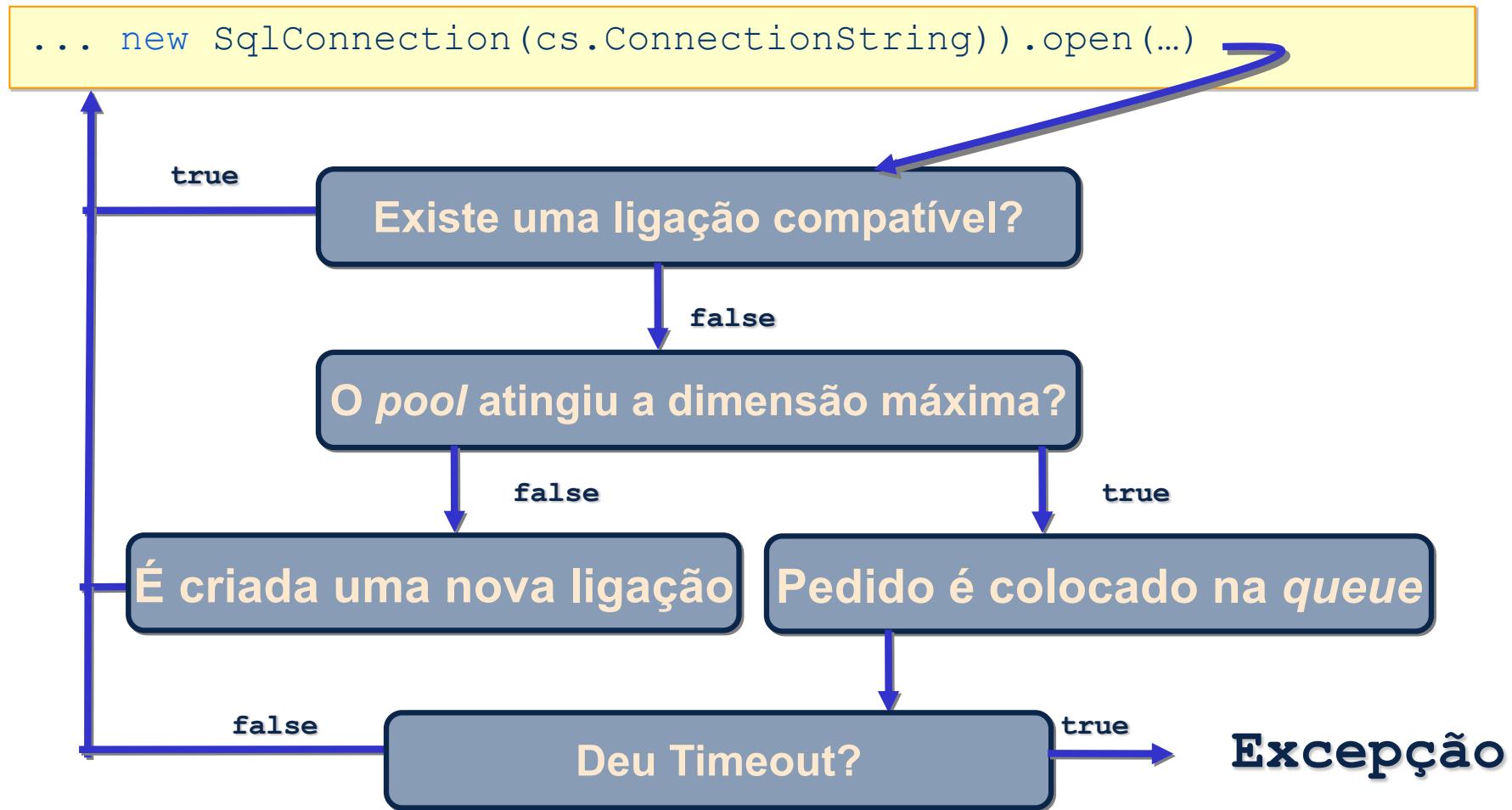
Connection Pooling

- No entanto, o *pool* só é criado quando a primeira ligação é aberta
- As ligações não ficam indefinidamente no *pool*
- Quando superam um tempo de inactividade (de 4 a 8 minutos) são removidas até o valor mínimo de ligações a deixar no *pool*



Connection Pooling – Diagrama de acções

- Como se processa a gestão de ligações (admitindo a existência de um pool)...



Connection Pooling – Pool corrompido

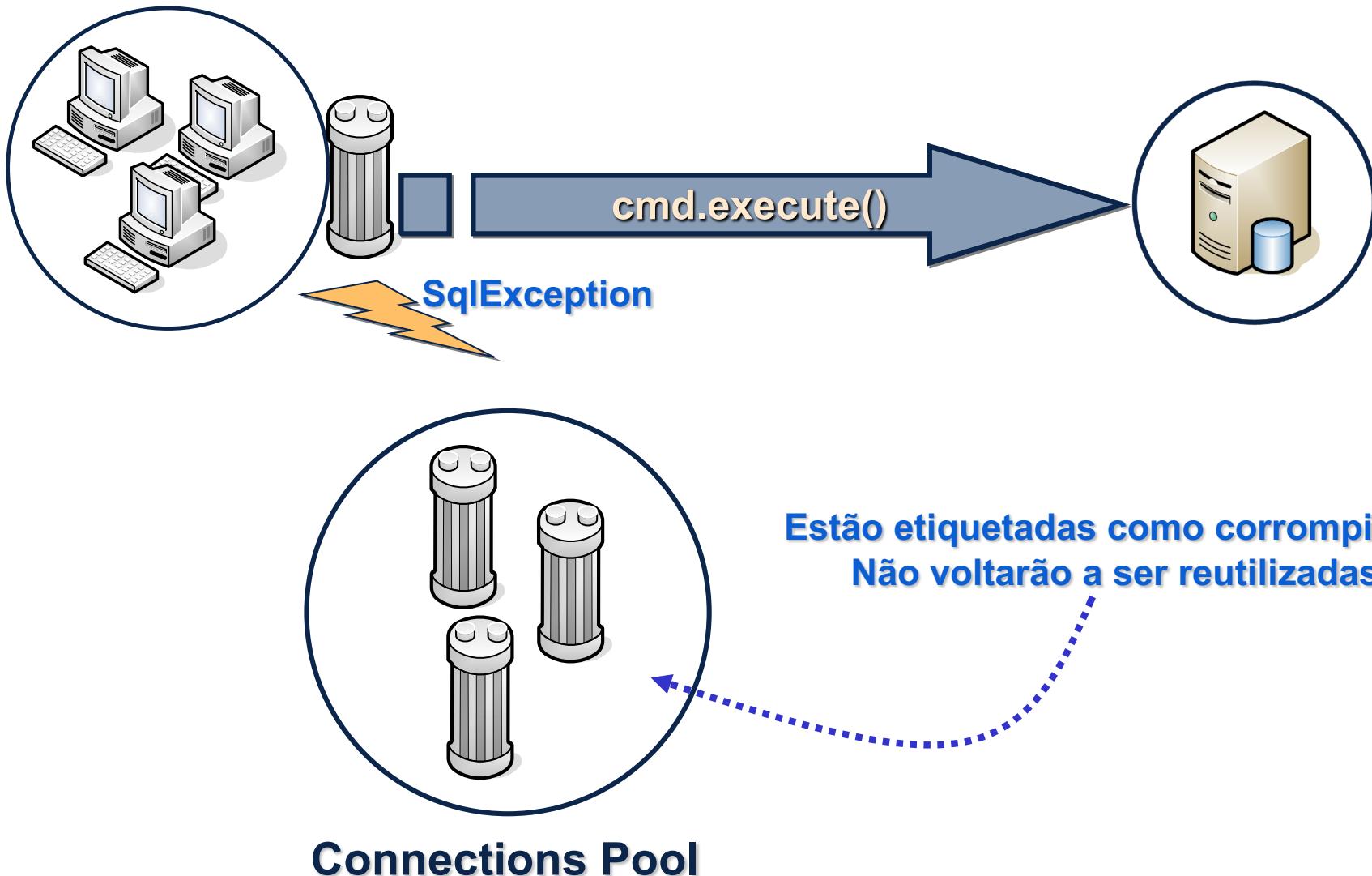
- Imagine-se o seguinte cenário:
 - Abrem-se três ligações
 - Desliga-se e reinicia-se o processo do SQL server
 - As ligações do *pool* estão corrompidas
 - “Abre-se” uma nova ligação: Tudo corre bem, pois existem três ligações disponíveis. Uma delas é devolvida para ser reutilizada
 - Executa-se um comando e...
 - É lançada uma exceção

A transport-level error has occurred when sending the request to the server.
(provider: Shared Memory Provider, error: 0 - No process is on the other end of
the pipe)

- E as restantes ligações? Estão igualmente corrompidas!

Connection Pooling – Pool corrompido

ADO.NET > 2.0



Connection Pooling – Pool corrompido

- O ADO.NET já resolve (parcialmente) este problema, quando detecta certos tipos de excepções, que indicam que o *pool* está corrompido
- Assim todas as ligações existentes no *pool* são etiquetadas como corrompidas
- Assim, da próxima vez que alguém tentar estabelecer uma ligação, é feita uma tentativa para abrir uma nova: Nunca é retirada do *pool* uma ligação etiquetada como corrompida!
- Esta acção tem efeito apenas no *pool* que originou a excepção e não nos restantes

Connection Pooling – Limpar o pool

- Existem agora dois métodos disponíveis nas classes SqlConnection e OracleConnection
 - ClearPool(connection) e ClearAllPools()
- Estes métodos permitem ao programador etiquetar todas as ligações (de um *pool* específico ou de todos os *pools*) como corrompidas
- Estes métodos não fecham ligações nem as retiram do *pool*. Apenas garantem que não são reutilizadas futuramente

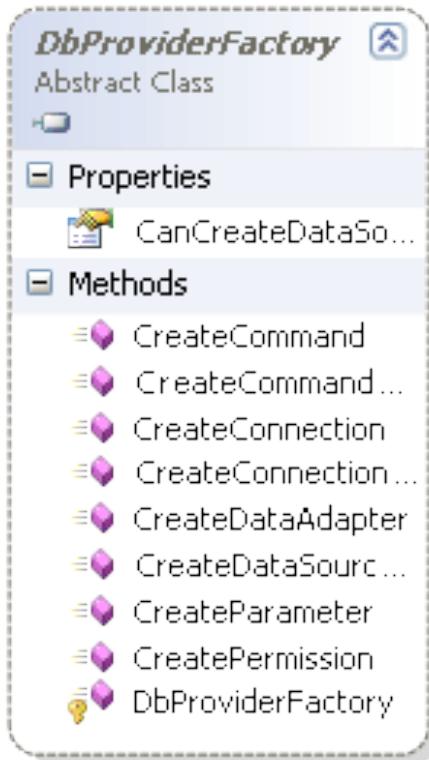
```
...
catch (SqlException ex) {
    if ( ! fatalNetworkException(ex.Number) ) throw ex;
    SqlConnection.ClearAllPools();
    //tenta-se nova ligação...
}
```

demo

System.Data.Common.DbProviderFactory

- O código anterior sofre de alguns problemas
 - A criação de uma ligação é feita com código específico para cada *provider*
 - Tem problemas de manutenção de código
 - Falta de generalidade
- A versão do ADO.NET introduziu alterações ao nível da estrutura de classes, alterações que contemplaram um modelo de suporte para código independente do *provider*
- Foi introduzida a classe `DbProviderFactory`
- Como em casos anteriores, existe uma implementação específica para cada .NET *provider*

System.Data.Common.DbProviderFactory



- Esta classe dispõem dos métodos que permitem a criação de instâncias específicas de cada um dos objectos fundamentais para a execução de comandos
- No entanto, falta ainda um passo fundamental: a obtenção de uma implementação específica de `DbProviderFactory`
- Esse serviço é disponibilizado pela classe estática `DbProviderFactories`
- Através do método `DbProviderFactories.GetProvider(name)` é possível obter, através do nome, um `DbProviderFactory` específico.

```
DbProviderFactory fact =  
    DbProviderFactories.GetFactory("System.Data.SqlClient")
```

Manipular a lista de .NET providers

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.data>
    <DbProviderFactories>
      <remove invariant="System.Data.Odbc"/>
      <remove invariant="System.Data.OleDb"/>
    </DbProviderFactories>
  </system.data>
...

```

Remove da lista de providers disponíveis, o ODBC e o OleDb

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.data>
    <DbProviderFactories>
      <clear/>
      <add name="SqlClient Data Provider"
        invariant="System.Data.SqlClient"
        description=".Net Framework Data Provider for SqlServer"
        type="System.Data.SqlClient.SqlClientFactory, System.Data,
        Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089" />
    </DbProviderFactories>
  </system.data>
...

```

Limpa a lista que se encontra em machine.config

Disponibiliza apenas o provider System.Data.SqlClient

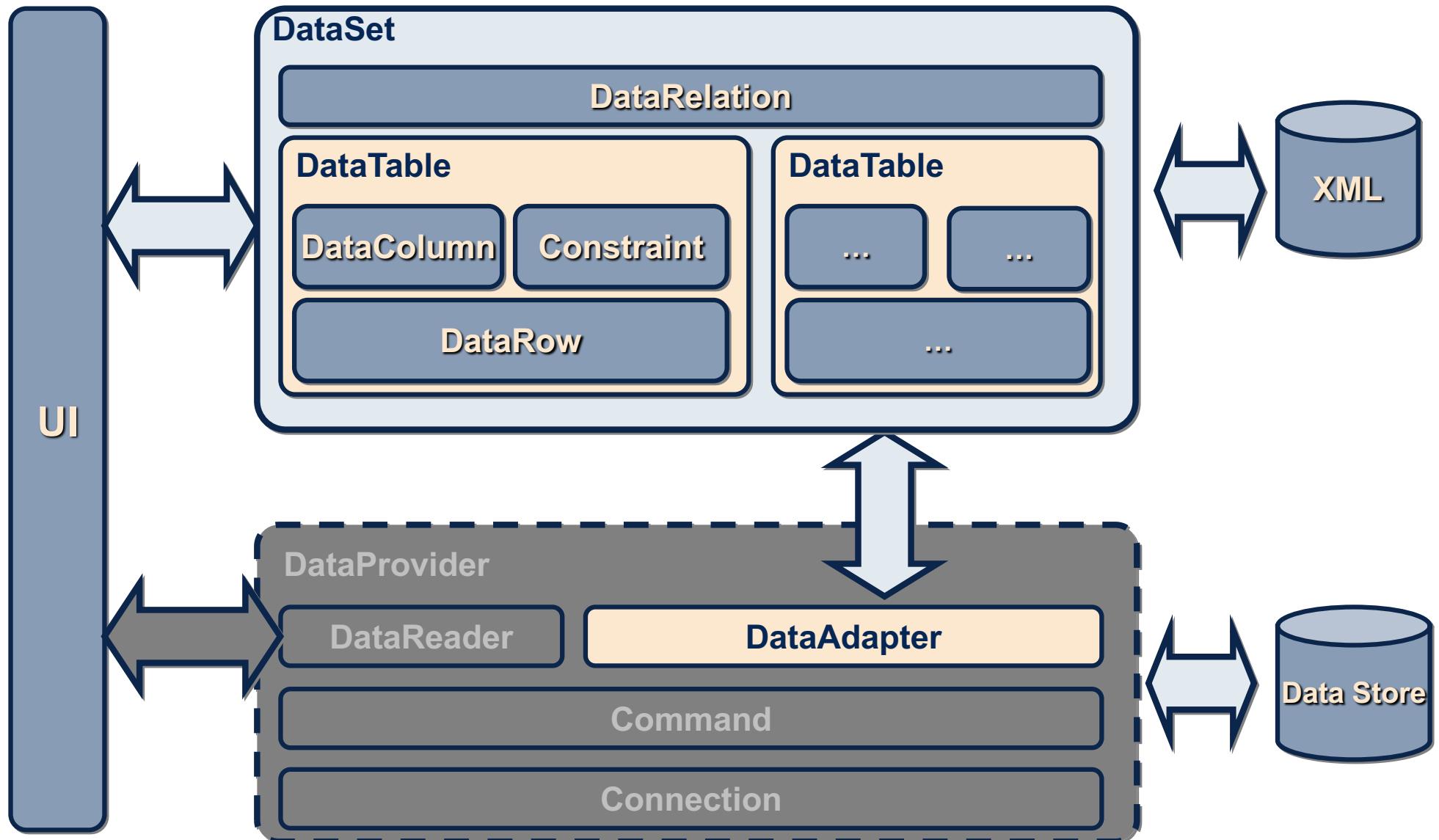
Código independente do *Provider*

```
DbProviderFactory fact = DbProviderFactories.GetFactory(  
    (string)ConfigurationManager.AppSettings["provider"]);  
using (DbConnection con = fact.CreateConnection())  
{  
    string  
    cs= ConfigurationManager.ConnectionStrings["cs"].ConnectionString;  
    con.ConnectionString = cs;  
    using (DbCommand cmd = con.CreateCommand())  
    {  
        cmd.CommandText = "select @@version";  
        con.Open();  
        string ver = (string)cmd.ExecuteScalar();  
        MessageBox.Show(ver);  
    }  
}
```

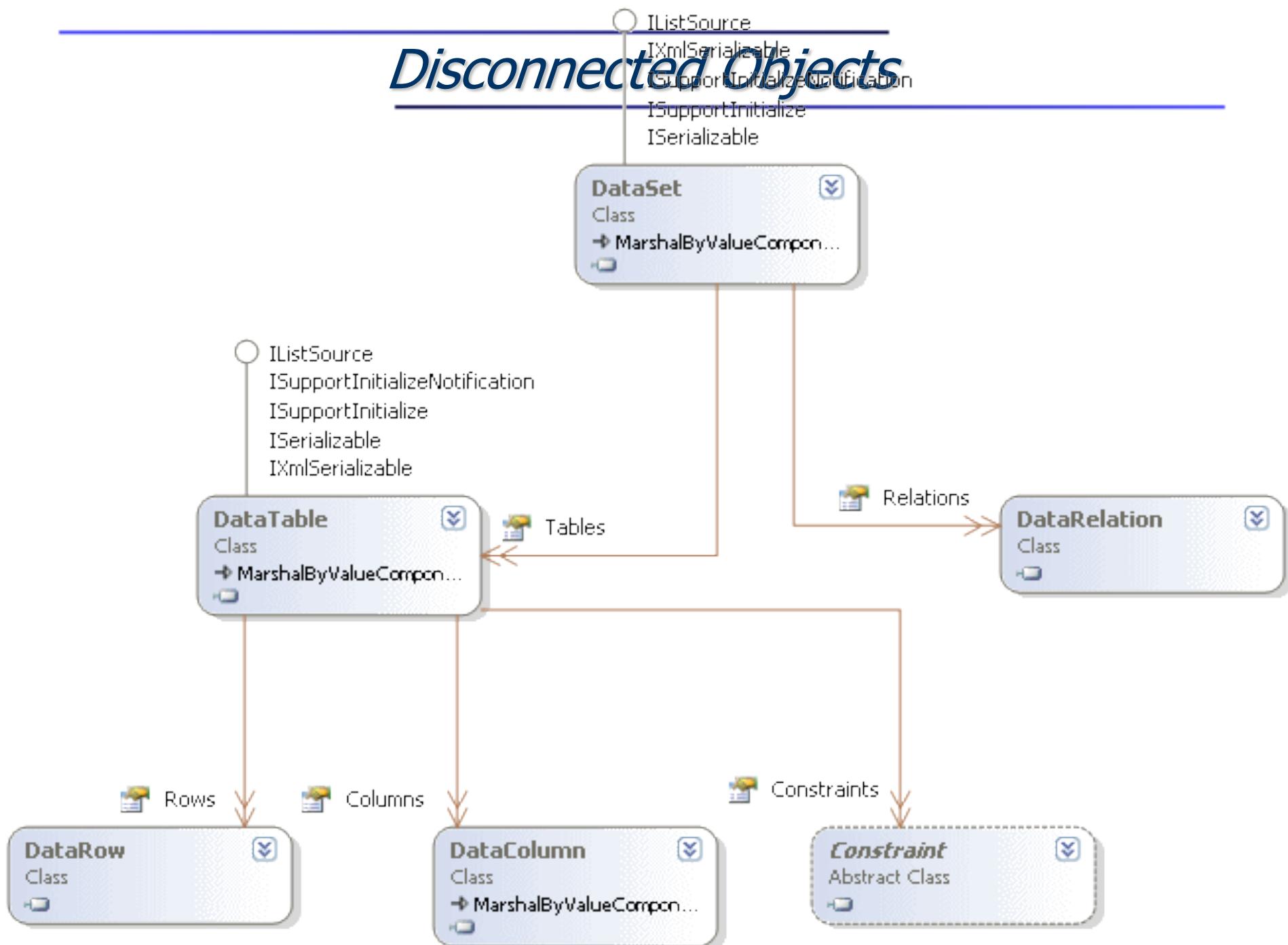
App.config

```
<appSettings>  
    <add key="provider" value="System.Data.SqlClient"/>  
</appSettings>
```

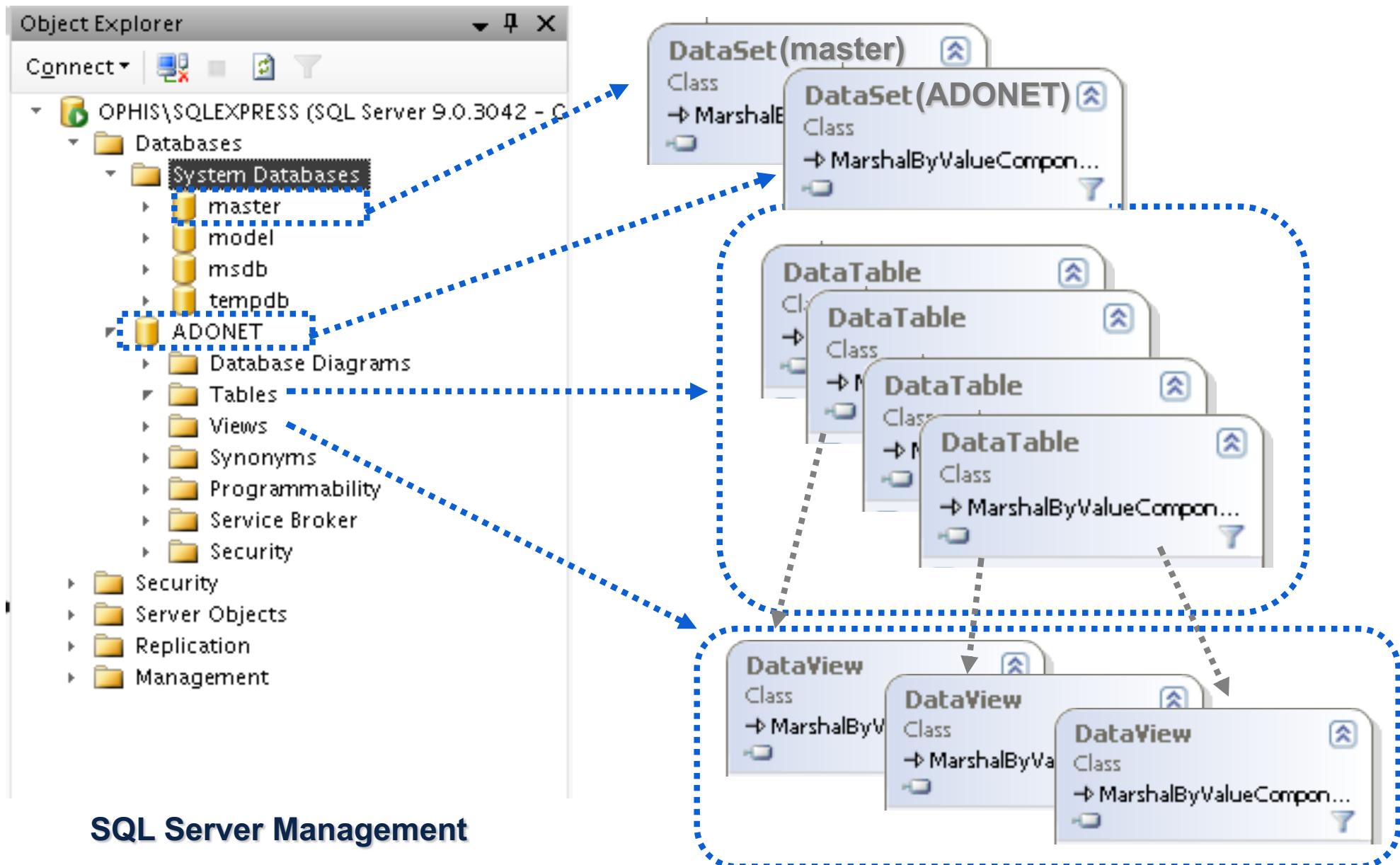
ADO.NET



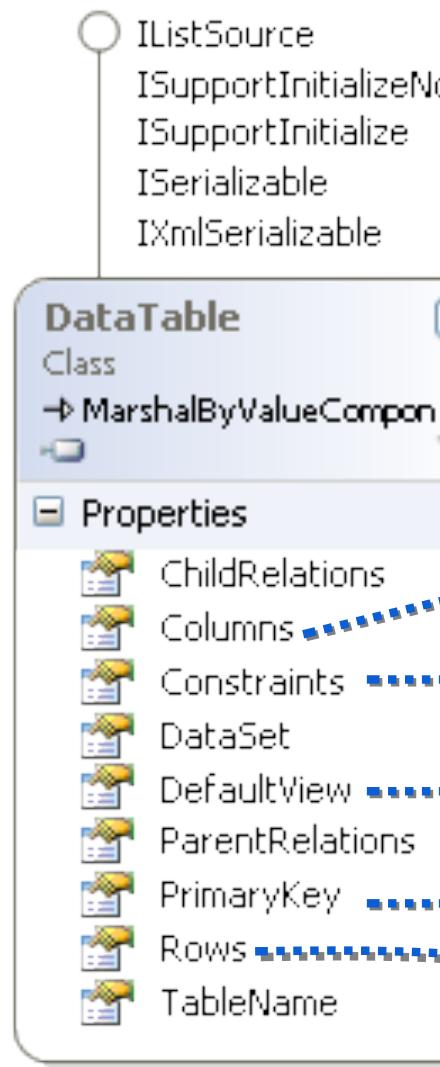
Disconnected Objects



Disconnected Objects <-> SGBD



System.Data.DataTable



- Um `DataTable` representa dados em forma tabular, residentes em memória
- Para ter dados *offline*, é necessário ter pelo menos, um `DataTable`

Array de DataColumns – O Esquema

Array de Constraints – As restrições

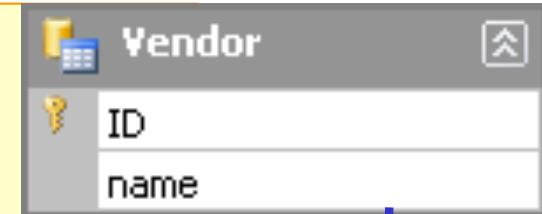
Vista sobre a tabela - `DataView`

Chave primária – `DataColumn` []

Array de DataRows – Os tuplos

System.Data.DataTable

```
DataTable vendor = new DataTable("Vendor");
vendor.Columns.Add("Id", typeof(Int32));
vendor.Columns.Add("name", typeof(String));
vendor.PrimaryKey = new DataColumn[]
{ vendor.Columns["Id"] };
```



Preencher

```
DataRow r1 = vendor.NewRow();
r1["Id"] = 1; r1["name"] = "teste1";
vendor.Rows.Add(r1);

DataRow r2 = vendor.NewRow();
r2["Id"] = 2; r2["name"] = "teste2";
vendor.Rows.Add(r2)
```

↓
Listar

```
foreach (DataRow r in vendor.Rows)
    Console.WriteLine("{0}\t|{1}", r["Id"], r["name"]);
```

1	teste1
2	teste2

System.Data.DataTable <-> SGBD

Programaticamente

```
DataTable vendor = new  
    DataTable("Vendor");  
  
vendor.Columns.Add("Id", typeof(Int32));  
...
```

```
DataRow r1 = vendor.NewRow();  
r1["Id"] = 1; r1["name"] = "teste1";  
vendor.Rows.Add(r1);  
...
```

```
foreach (DataRow r in vendor.Rows)  
    Console.WriteLine("{0}\t| {1}",  
        r["Id"], r["name"]);
```

Reside em memória !

SQL

```
CREATE TABLE vendor  
(  
    id int primary key,  
    name varchar(50)  
)
```

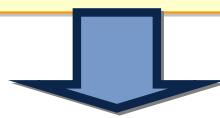
```
INSERT INTO vendor  
VALUES(1,'teste1')
```

```
SELECT id, name  
FROM Vendor
```

Reside no SGBD

System.Data.DataTable – Tipificado

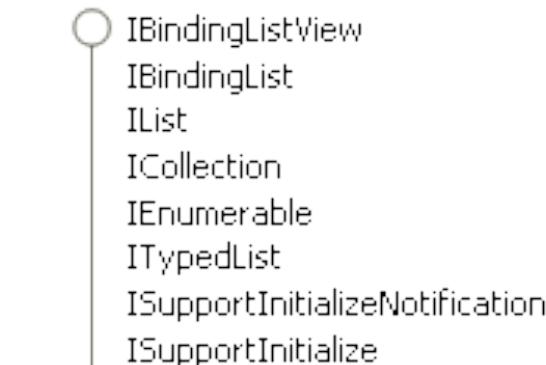
```
class VendorDataTable : DataTable
{
    public VendorDataTable() : base("Vendor")
    {
        this.Columns.Add("ID", typeof(Int32));
        this.Columns.Add("name", typeof(String));
        this.PrimaryKey=new
        DataColumn[] {this.Columns["ID"]};
    }
    public DataColumn ID { get { return this.Columns["Id"]; } }
    public DataColumn Name{ get { return this.Columns ["Name"]; } }
}
```



Vantagens

- Validação de tipos em tempo de compilação;
- Facilita a identificação de erros de cada vez que existe uma actualização do esquema e consequente actualização da definição do DataTable

System.Data.DataView



Properties	
AllowDelete	
AllowEdit	
AllowNew	
Count	
Item	
RowFilter	
Sort	
Table	
Methods	
ToTable	(+ 3 over...)

- Uma instância de **DataView** não contém dados; consiste apenas numa vista para os dados armazenados numa **DataTable**
- Por omissão, uma instância de **DataTable** já tem uma **DataView**, acessível através da propriedade **DefaultView**.

Filtro: Colocar a expressão de condição, de forma análoga a uma condição em SQL

Permite ordenar os tuplos. Sintaxe semelhante à utilizada em SQL

Permite Criar uma tabela a partir da **DataView**

System.Data.DataView

```
DataView view = vendor.DefaultView;
```



Filtrar

```
view.RowFilter = " Id <> 2";
```



Ordenar

```
SELECT *  
FROM Vendor  
WHERE Id <> 2  
ORDER BY name DESC
```

```
view.Sort = "name DESC";
```

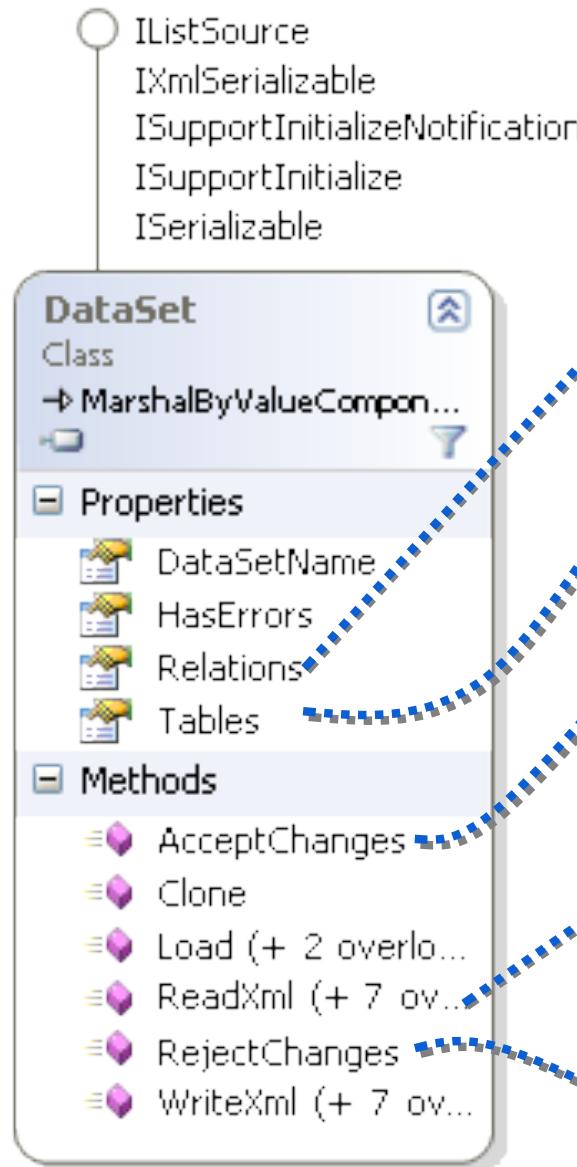


Listar

```
DataTable t = view.ToTable();  
foreach (DataRow r in t.Rows)  
    Console.WriteLine("{0}\t|{1}", r["Id"], r["name"]);
```

```
1 | teste1
```

System.Data.Dataset



- Um **DataSet** pode ser visto como uma representação, em memória, de um modelo relacional

Conjunto de DataRelation. Contém as relações entre as diversas DataTables que constituem o DataSet

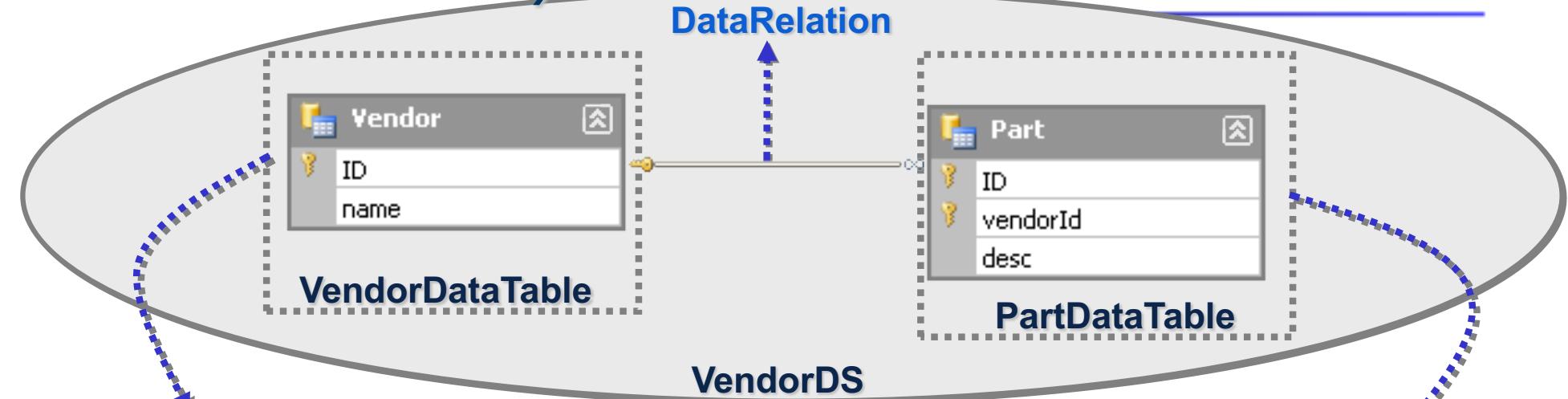
As DataTables que constituem o DataSet

Permite indicar que todas as alterações são aceites, ajustando o estado dos DataRow's de todas as DataTables do DataSet

Os métodos Read/WriteXml permitem ler/escrever DataSets inteiros de/em ficheiros XML

Permite anular todas as alterações efectuadas até à última chamada do **AcceptChanges**

System.Data.DataSet



```
class VendorDataTable : DataTable
{ ... }

class PartDataTable : DataTable
{
    public PartDataTable(): base("Part") {
        this.Columns.Add("ID", typeof(Int32));
        ...
        this.PrimaryKey = new DataColumn[]
        { this.Columns["ID"], this.Columns["vendorId"] };
    }
    ...
}
```

System.Data.DataSet

```
class VendorDataSet : DataSet  
{  
    DataTables  
    PartDataTable part;  
    VendorDataTable vendor;  
    public VendorDataSet() : base ("VendorDs")  
    {  
        part = new PartDataTable();  
        vendor = new VendorDataTable();  
        this.Tables.Add(vendor);  
        this.Tables.Add(part);  
        this.Relations.Add("fk_vendor_part",  
                           vendor.ID, part.VendorId, true);  
    }  
    ...  
}
```

DataRelation

Nome do DataSet

Nome da relação

DataTables

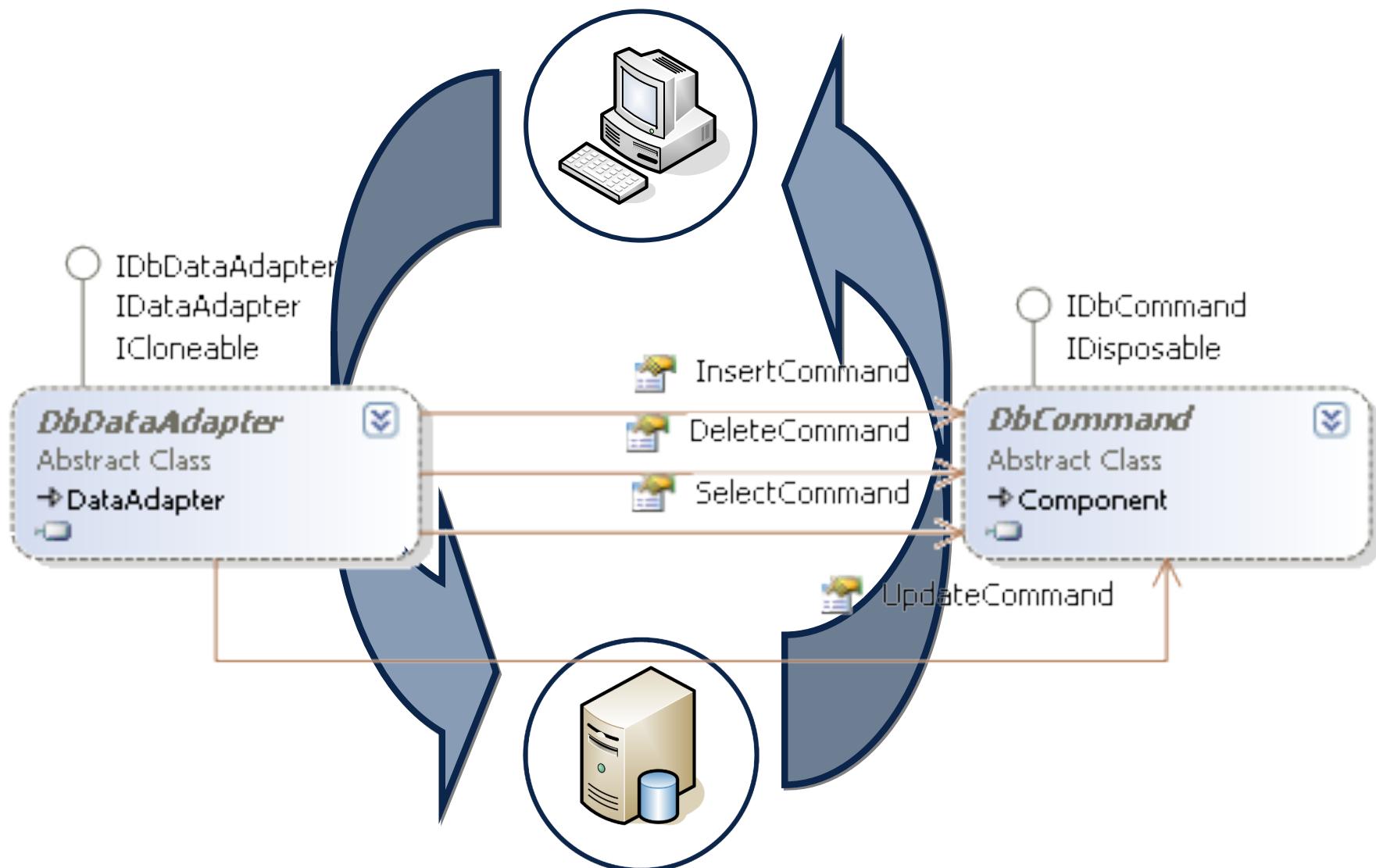
Parent (1)

Child (muitos)

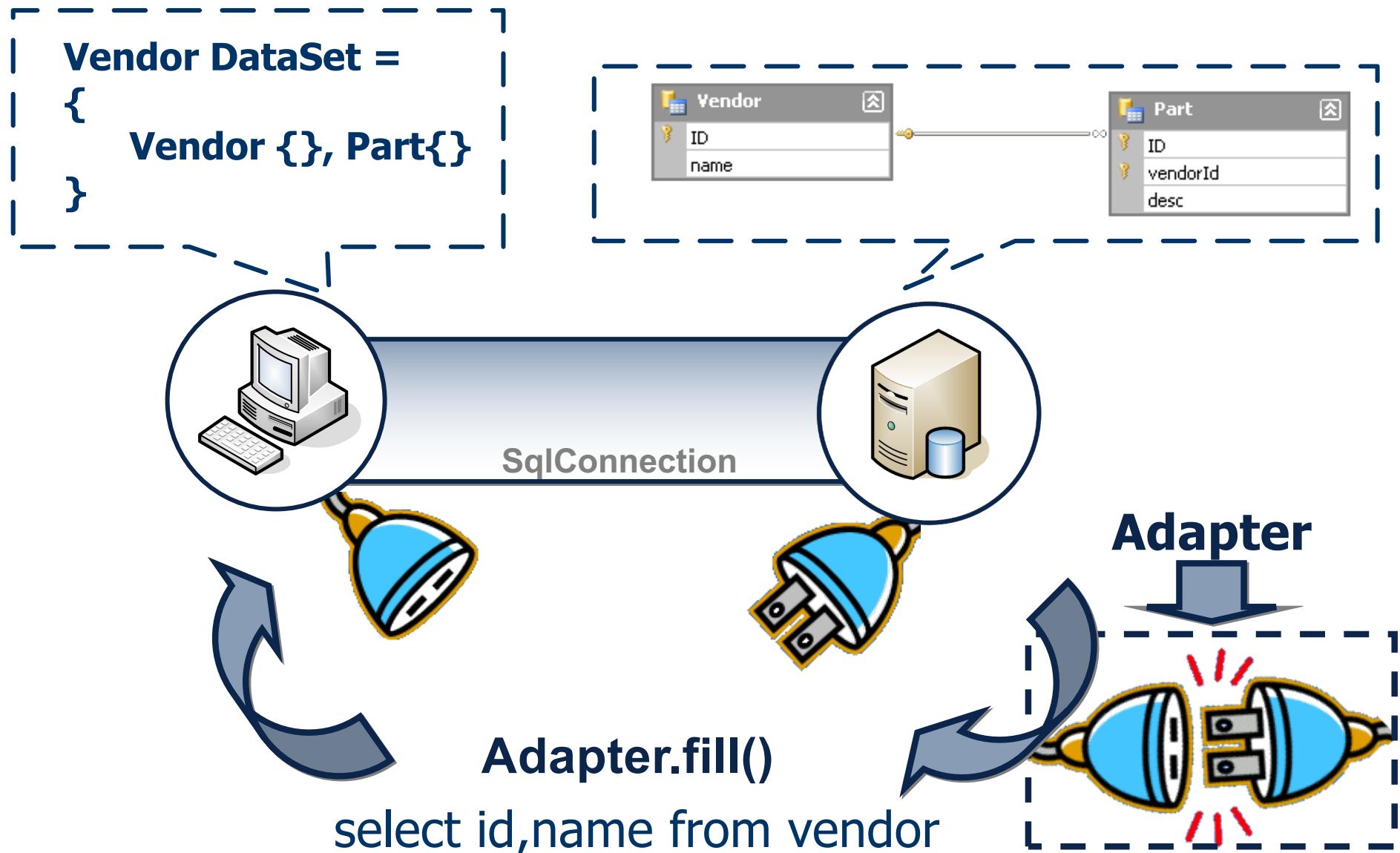
Se cria ou não uma restrição

System.Data.Common.DbDataAdapter

DataSet, DataTables, ...



System.Data.Common.DbDataAdapter



System.Data.Common.DbDataAdapter

```
VendorDataSet ds = new VendorDataSet(); ..... ➔ Criar o Dataset  
SqlDataAdapter adapterVendor,adapterPart;
```

Criar DataAdapters: BD ➔ VendorDataTable

```
cmd.CommandText = "select id,name from vendor";  
adapterVendor = new SqlDataAdapter(cmd);  
adapterVendor.Fill(ds.Vendor);
```

Ligaçāo à BD

Criar DataAdapters: BD ➔ PartDataTable

```
cmd.CommandText = "select id,descr,vendorId from part";  
adapterPart = new SqlDataAdapter(cmd);  
adapterPart.Fill(ds.Part);
```

System.Data.Common.DbDataAdapter

Iterar sobre os tuplos de VendorDataTable

```
foreach (DataRow r in ds.Vendor.Rows)
{
    Console.WriteLine("{0} | {1}", r[0], r[1]);
    foreach (DataRow child in r.GetChildRows(ds.VendorPartRelation))
        Console.WriteLine("\t{0} | {1} | {2}",
            child[0], child[1], child[2]);
}
```

Obter os tuplos de PartDataTable que têm uma chave estrangeira igual ao ID do tuplo corrente de VendorDataTable (r)

```
Select p.id, p.desc, p.vendorId
From Part p, Vendor r
where p.vendorId = r.id and r.id = ...
```

DataSet <> XML

- A informação presente num **DataSet** pode ser colocada num ficheiro XML

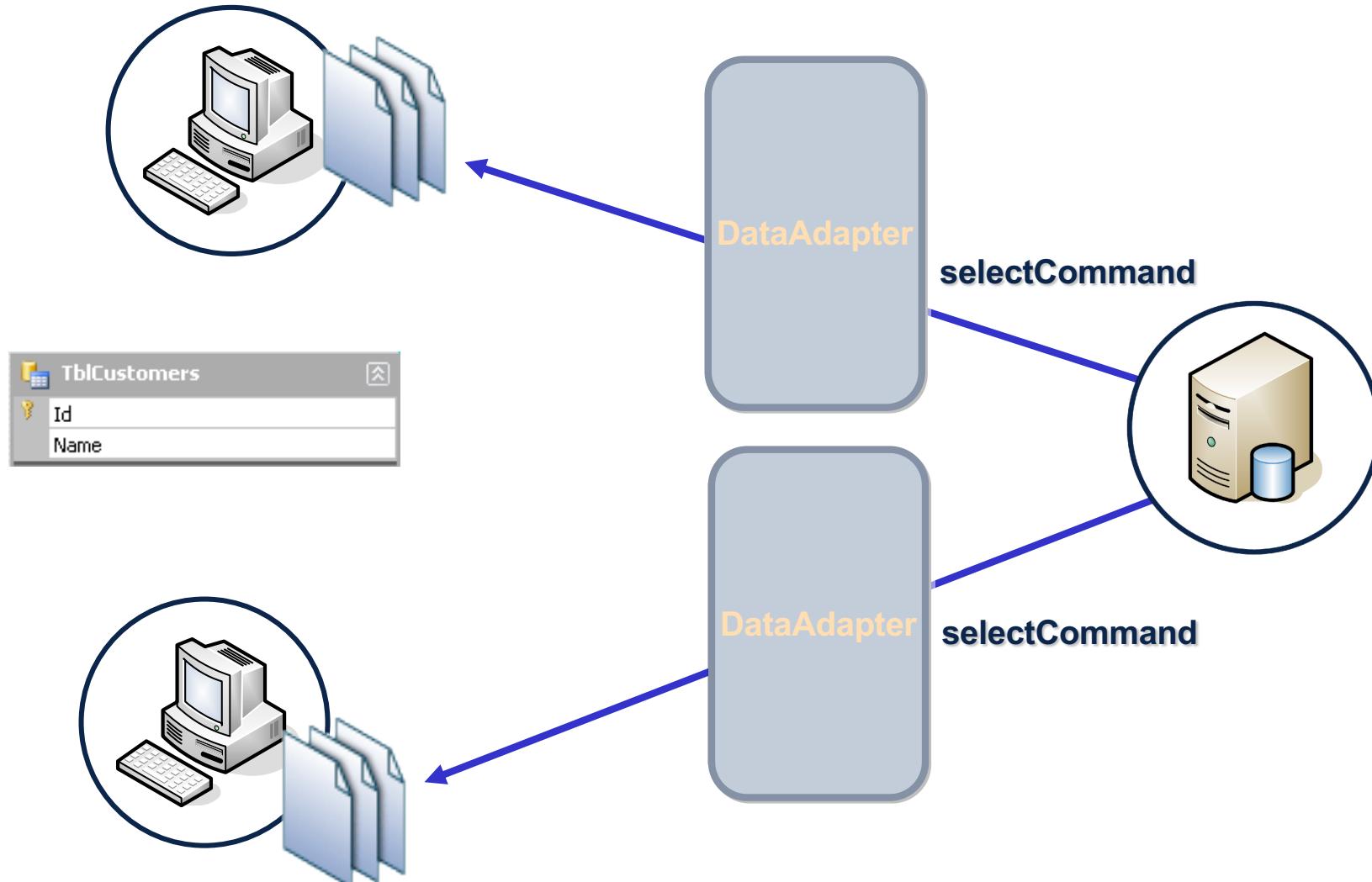
```
ds.ReadXml ("VendorDS.xml");
```

```
VendorDataSet ds = new VendorDataSet();  
...  
ds.WriteXml ("VendorDS.xml");
```

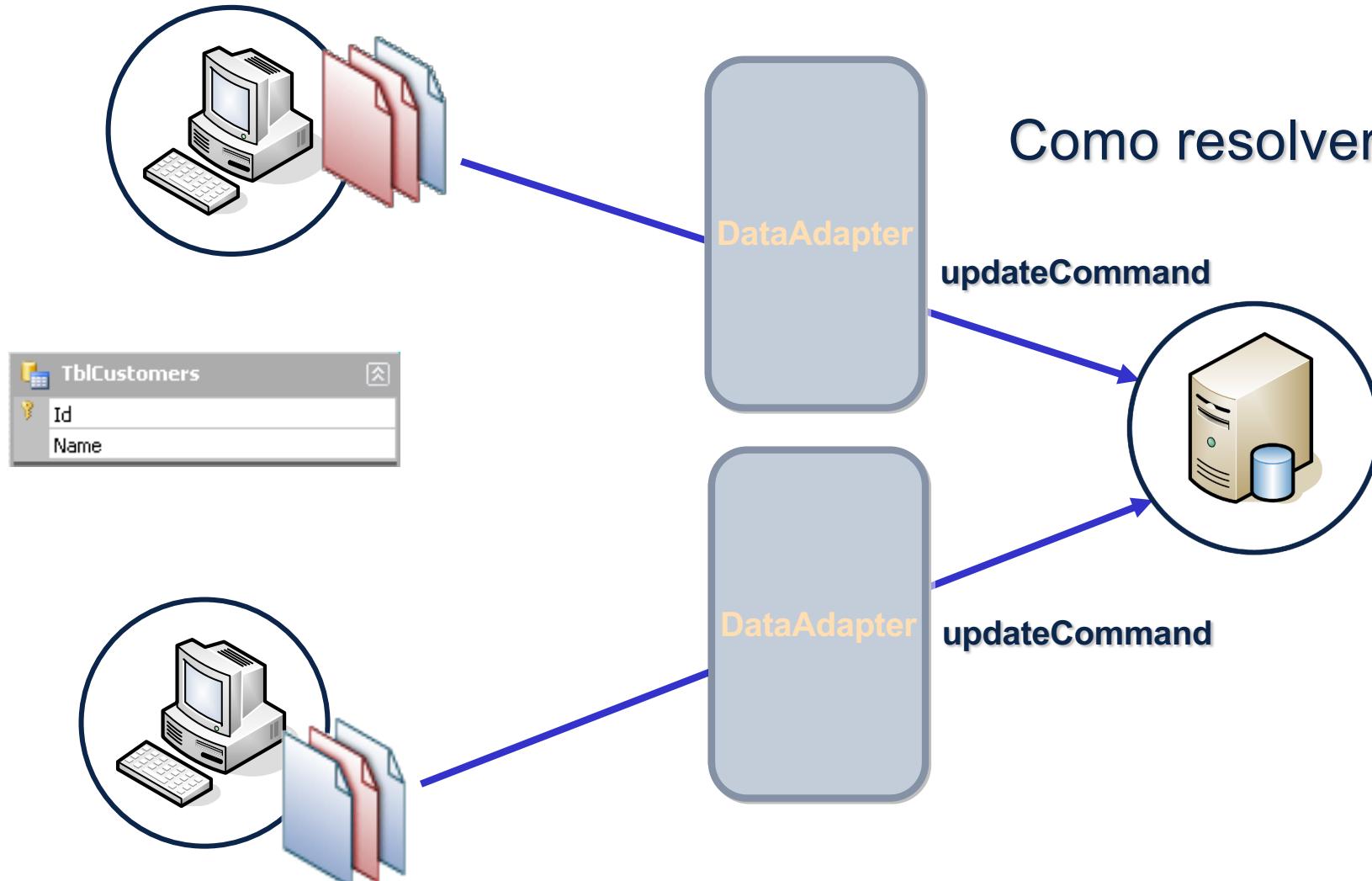
```
<?xml version="1.0" standalone="yes"?>  
<VendorDs>  
  <Vendor>  
    <ID>1</ID>  
    <name>Vendor 1</name>  
  </Vendor>  
  <Part>  
    <ID>1</ID>  
    <vendorId>1</vendorId>  
    <descr>Part 1</descr>  
  </Part>  
  <Part>  
    <ID>1</ID>  
    <vendorId>2</vendorId>  
    <descr>Part 1</descr>  
  </Part>  
</VendorDs>
```

- ReadXml: por omissão e caso o XML não inclua um *schema*, este método pode inferir o *schema* apropriado para esses dados

Conflitos



Conflitos



Como resolver o conflito?

Conflitos

- Uma vez que é possível alterar os dados de forma desligada do repositório da dados, é possível que dois utilizadores tenham alterado os mesmos dados
- Não é possível envolver todas as alterações numa transacção
- Têm de ser encontrados outros mecanismos de controlo de concorrência
- De acordo com a aplicação, devem ser tidos em conta os cenários de conflito e, resolvê-los programaticamente!
- No limite será utilizador a decidir

Possibilidades

- Prioridade no tempo - o primeiro ganha: Esta é uma resolução simples de implementar e aquela que é a implementada DbDataAdapter. É garantida pela inclusão de todos os campos na cláusula where do update.

```
UPDATE dbo.TblCustomers  
SET Id = @Id, Name = @Name  
WHERE (Id = @Original_id) AND (Name = @Original_Name)
```

- Prioridade no tempo - o último ganha: Também é simples de implementar. Basta deixar na cláusula where apenas a chave primária da tabela.

```
UPDATE dbo.TblCustomers  
SET Name = @Name  
WHERE Id = @Original_id
```

Possibilidades

- Prioridade pelo tipo de utilizador – Os conflitos são geridos de acordo com o tipo de utilizador. Admitindo dois papéis, administradores e vendedores, pode-se dar prioridade aos primeiros. Quando o tipo de utilizador é igual, implementa-se um dos esquemas anteriores.
- Prioridade à localização – O *updates* são geridos de acordo com o local de onde veio. Admitindo os mesmos papéis anteriores, pode dar-se prioridade aos vendedores sobre determinadas tabelas (e.g. vendas). Também aqui, poder-se-á ter que utilizar um dos esquemas anteriores, quando a localização é a mesma.
- Utilizador têm última palavra

Criar uma transacção - *DbTransaction*

- Em ADO.NET uma transacção está intimamente associada com uma ligação
- É através desta, que pode ser criado um contexto transaccional que irá envolver todo o trabalho efectuado

```
SqlConnection con = new SqlConnection();  
SqlTransaction tran = con.BeginTransaction();
```

- Pode-se especificar na sua criação o nível de isolamento desejado

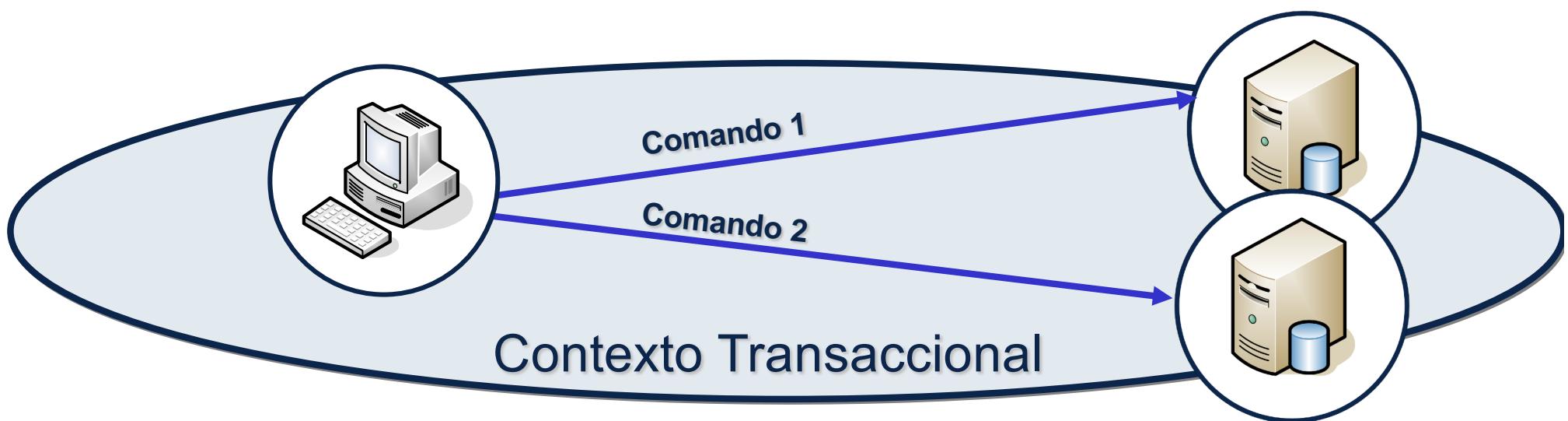
```
SqlTransaction tran =  
con.BeginTransaction(  
System.Data.IsolationLevel.RepeatableRead);
```

DbTransaction

```
using (SqlConnection con = new SqlConnection())  
{  
    con.ConnectionString = cs;  
    con.Open();  
    SqlTransaction tran = con.BeginTransaction();  
    try  
    {  
        SqlCommand cmd = con.CreateCommand();  
        cmd.Transaction = tran;  
        cmd.CommandText = " raiserror ('simulacao erro',18,1)";  
        tran.Commit();  
    }  
    catch (Exception ex)  
    {  
        tran.Rollback();  
    }  
}
```

System.Transactions

- É possível utilizar as transacções disponibilizadas pela plataforma .NET e gerir contextos transaccionais conjuntamente com o SQL Server
- O modelo de programação torna-se mais simples e transparente
- Principalmente quando no código se usam várias ligações a repositórios de dados, onde se quer manter um contexto transaccional



Transacções locais e distribuídas

- **Transacções locais:** Utiliza um gestor transaccional (e.g. SQL Server). Desde que apenas seja utilizada uma ligação, uma transacção é local
- **Transacções distribuídas:** Dois ou mais gestores transaccionais entram no contexto transaccional, sendo necessário utilizar o protocolo *2 Phase commit (2PC)*
- É neste último contexto que o suporte transaccional do .NET simplifica o trabalho do programador

System.Transactions.TransactionScope

- A classe `TransactionScope` pode ser utilizada para gerir o contexto transaccional
- Esta classe cria um tipo de transacção designada da "*local lightweight transaction*"
- Esta transacção é automaticamente promovida a distribuída se tal for necessário (apenas com o SQL server 2005)
- Note-se que esta transacção é uma transacção implícita, da qual apenas manipulamos um contexto transaccional
- O *commit* ou *rollback* é feito automaticamente, assim que o contexto termina

System.Transactions.TransactionScope

```
using (TransactionScope tran = new TransactionScope())
{
    using (SqlConnection con = new SqlConnection())
    {
        con.ConnectionString = cs;
        con.Open();

        SqlCommand cmd = con.CreateCommand();
        cmd.CommandText = " raiserror ('simulacao erro',18,1)";

        tran.Complete();

    }
}
```

Se existir uma excepção (e vai existir) o método complete nunca é evocado, originando o *rollback* da transacção

Modificar opções no contexto transaccional

- Quando o TransactionScope é criada é possível especificar opções sobre o contexto a criar e o tipo de isolamento pretendido

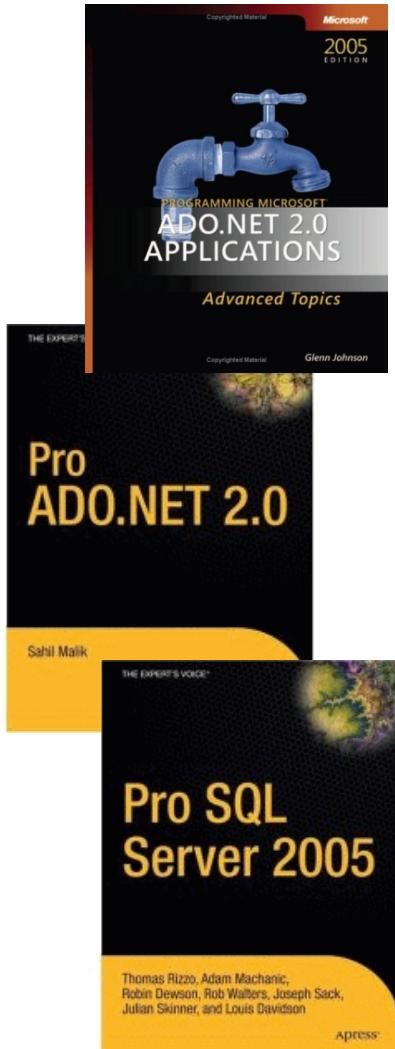
```
TransactionOptions opt = new TransactionOptions();  
opt.IsolationLevel=System.Transactions.IsolationLevel.RepeatableRead;  
using (TransactionScope tran = new  
    TransactionScope(TransactionScopeOption.Required, opt))  
{  
    ...  
}
```

- *Required*: É necessário uma transacção. Se não existe nenhuma a decorrer é criada uma nova. Caso contrário, é utilizada a existente
- *RequiresNew*: É sempre necessário a um nova transacção. Desta forma são isolados os desfechos das transacções.
- *Suppress*: Não necessita de uma transacção. É uma forma de correr código sem suporte transaccional no meio e uma transacção

LTM vs DTC

- Se for utilizado o `TransactionScope`, a transacção criada é local (preferencialmente) e gerida pelo LTM (*Lightweight Transaction Manager*)
- Note que o LTM precisa de um gestor transaccional que garanta as todas as propriedades ACID da transacção
- O seu papel é monitorar o desenrolar das accções
- Quando é utilizado outro gestor transaccional dentro do mesmo contexto, o LTM desencadeia o processo de promoção da transacção em curso para um transacção distribuída (se o gestor implementar `IPromotableSinglePhaseNotification`)
- A partir deste ponto, é o DTC (*Distributed Transaction Coordinator*) que gera o contexto transaccional

Referências



- **Glenn Johnson**
Publisher: Microsoft Press
Copyright: 2005
- **Sahil Malik**
Publisher: Apress
Copyright: 2005
- **Thomas Rizzo**
Publisher: Apress
Copyright: 2005