# 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
* **Connected Classes**
  + *Provider-Independent* APIs
  + Controlo do *Connection pooling*
  + Integração com o namespace System.Transactions e promoção automática de transações
  + 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 interação já conhecido de outras tecnologias de acesso a dados

Parameters

Connection

Transaction

## .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
  + OdbcXxxx (e.g. OdbcConnection e OdbcCommand )
  + OleDBXxxx
  + SqlXxxx
  + OracleXxx

# *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 transação – *BeginTransaction*
  + pode ser criado um comando – *CreateCommand*
* Isto espelha o facto de uma ligação ser o ponto central e unificador das classes DbXxxx
* *IDbConnection* deriva de *IDisposable*, por forma a possibilitar a libertação de recursos pela infraestrutura .NET

### Execução de um comando: Sequência típica de ações

1. Estabelecer uma ligação à fonte de dados
2. Especificar um comando a ser efetuado
   * + Opcionalmente pode ser necessário utilizar parâmetros
3. Se necessário manipular transações
4. Executar o comando
5. Caso o comando produza um conjunto de dados, manipular o resultado
6. Se existir um contexto transacional ativo, indicar o sucesso da ação
7. Fechar a ligação

A picture containing timeline

Description automatically generated

*Erros:*

* dr não foi fechado
* sqlCommand não foi fechado
* fiz close da ligação (é preciso fazer Dispose)

## *ConnectionString*

* A *ConnectionString* é específica para cada provider
* Regra geral, é necessário especificar sempre
  + **A fonte de dados**
  + **Informação relativa ao utilizador**

con.ConnectionString= @"Data Source=.\SQLEXPRESS;

Initial Catalog=ADONET;Integrated Security=True";

* 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 *connectionsStrings* 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

A picture containing graphical user interface

Description automatically generated

Graphical user interface, text, website

Description automatically generated

* 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

## Fechar Ligações – Uma melhor abordagem

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

Graphical user interface, text, email

Description automatically generated

**Text

Description automatically generated with medium confidence**

## System.Data.Common.IDbCommand

* **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

Graphical user interface, application

Description automatically generated

### 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

Table

Description automatically generated

## 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

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

SOLUÇÃO:

## System.Data.Common.DbParameters

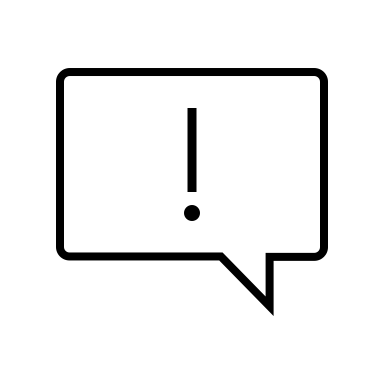
* Muitas das verificações podem ser feitas do lado do cliente, evitando uma abertura de uma ligação para um comando que nunca será possível executar
* Para os utilizar:
  + Os parâmetros têm de ter um nome. Este têm de começar por @
  + É através do nome que se acede ao valor do parâmetro
  + Têm de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

Notas:

* Manipular datas como carateres, caso tenhamos dificuldade em lidar com datetimes
* Resolve os ataques por sql injections
* Tipo de comando procedimento armazenado => usamos o nome do procedimento armazenado
* DbDataReader
* IDataRecord é importante
* ConnectionPooling
* Pq é que existem pools de objetos? Para reutilizar, diminuir o overhead da criação de objetos

Graphical user interface, timeline

Description automatically generated

O que temos de seguir

### Diferentes tipos de comandos

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

Table

Description automatically generated

## 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 objetos 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)
* No entanto o *indexer* retorna o valor de um campo como um *object*

A picture containing text

Description automatically generated

* 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

A picture containing chart

Description automatically generated

## Connection Pooling

* Com *Connection* *pooling*, o sistema pode gerir de forma eficiente as ligações, reutilizando as que estão ativas, em vez de serem criadas novas
* Para isso, existe um gestor de ligações, responsável por manter uma “lista” de ligações prontas a utilizar
* Para que o *pooling* funcione é necessário:
  + Que cada um dos caracteres da *connectionString* sejam exatamente 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
* 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

Graphical user interface, text, table

Description automatically generated

Notas:

* ConnectionTimeout -> quanto tempo vou estar a espera para se ligar ao servidor
* minPoolSize:10 -> não cria ligações nenhumas
* ConnectionPool é por processo

### 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
  + connectionPoolGroup
* 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
* Como se processa a gestão de ligações (admitindo a existência de um pool)…

Graphical user interface, text

Description automatically generated

## Connection Pooling – Pool corrompido

* 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

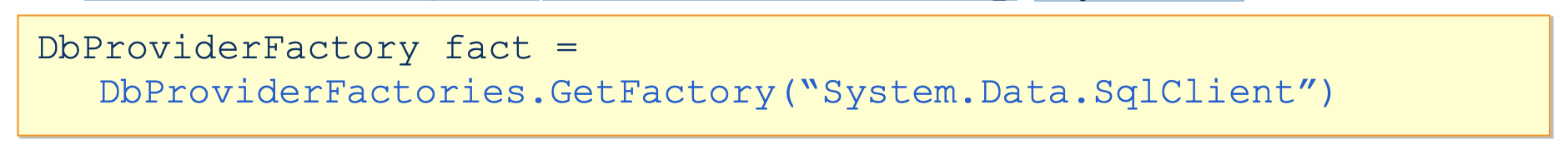
## Connection Pooling – Limpar o pool

* 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 futuramenteGraphical user interface, text, application

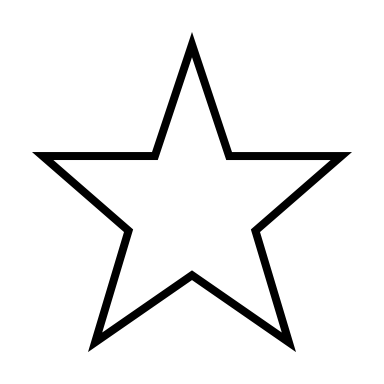
  Description automatically generated

## System.Data.Common.DbProviderFactory

* 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
* Esta classe dispõem dos métodos que permitem a criação de instâncias especificas de cada um dos objectos fundamentais para a execução de
* comandos
* No entanto, falta ainda um passo fundamental: a obtenção de uma 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.



## Código independente do Provider

Text

Description automatically generated

-- passar para slides pagina 78

## Criar uma transacção – DbTransactioN

* Em ADO.NET uma transacção está intimamente associada comuma ligação
* É através desta, que pode ser criado um contexto transacional que irá envolver todo o trabalho efectuado
* Pode-se especificar na sua criação o nível de isolamento desejado

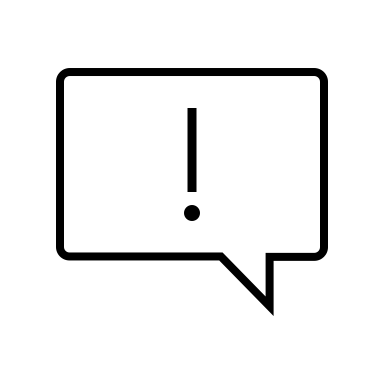
Graphical user interface, text

Description automatically generated with medium confidence

## DbTransaction

Graphical user interface

Description automatically generated with medium confidence

Não utilizar este código

## System.Transactions

* É possível utilizar as transacções disponibilizadas pelaplataforma .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 transacional

## 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 transacionais 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

Graphical user interface, text, application, email

Description automatically generated

## 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

Text

Description automatically generated with medium confidence

* 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 sãoisolados os desfechos das transacções.
* Suppress: Não necessita de uma transacção. É uma forma de correr códigosem suporte transaccional no meio e uma transacção

## LTM vs DTC

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