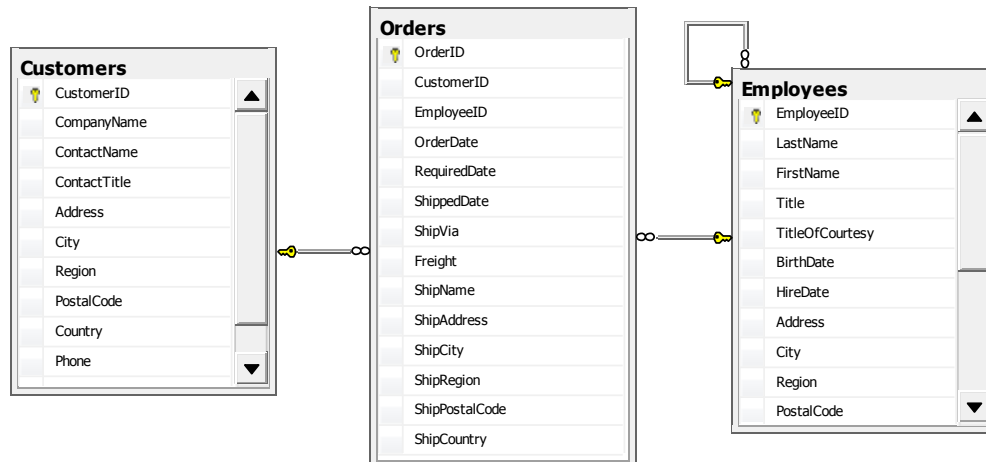


O objectivo deste trabalho é desenvolver uma *framework* SqlMapper capaz de criar uma instância de uma implementação de IDataMapper para uma determinada *entidade de domínio* (ED) tirando partido do serviço de reflexão (System.Reflection).

Para tal, pressupõe-se que cada ED tem uma tabela correspondente numa base de dados relacional.

A implementação do trabalho deve ser validada no **MÍNIMO** com testes unitários para as EDs correspondentes às seguintes tabelas (contudo a *framework* suportará a geração de *data mappers* para qualquer tabela além destas).



Considere o seguinte exemplo de criação de um *data mapper* para a ED Product (com uma tabela correspondente Products) através da classe Builder integrante da *framework* SqlMapper:

```
Builder b = new Builder(..., ...);
IDataMapper<Product> prodMapper = b.Build<Product>();
IEnumerable<Product> prods = prodMapper.GetAll();
```

```
public class Product {
    public int ProductID { set; get; }
    public string ProductName { set; get; }
    public string QuantityPerUnit { set; get; }
    public decimal UnitPrice { set; get; }
    public short UnitsInStock { set; get; }
    public short UnitsOnOrder { set; get; }
}
```

A interface IDataMapper obedece à seguinte definição, onde T representa o tipo da ED.

```
public interface IDataMapper<T>{
    IEnumerable<T> GetAll(); // Devolve todos os elementos da tabela correspondente
    void Update(T val); // Actualiza a linha que tem PK igual à propriedade PK de val (1er cap. Requisitos)
    void Delete(T val); // Apaga a linha com PK igual à propriedade PK de val
    void Insert(T val); // Insere uma nova linha com os valores de val e actualiza val com a PK devolvida
}
```

Figura 1

A *framework* SqlMapper deve ser implementada em **três partes, incrementalmente**.

- Leia ATENTAMENTE TODOS os requisitos incluindo os TRANSVERSSAIS às 3 partes
- NÃO implemente SIMULTANEAMENTE 2 alíneas
- Passe à alínea seguinte APENAS quando tiver CONCLUÍDA a alínea anterior
- A implementação de cada alínea deve ser validada com testes unitários que confirmem a sua correcção.
- Exige-se a resolução das alíneas opcionais para ter avaliação no trabalho acima de 16 valores.
- **DATA LIMITE DE ENTREGA: 30 de Junho de 2014**, incluindo relatório.

## 1ª parte: sem associação entre EDs.

1. Implemente a *framework* de acordo com a especificação da interface IMapper da Figura 1.
2. Altere a *framework* SqlMapper de modo a suportar o encadeamento de cláusulas de Where sobre o resultado de um GetAll. Para tal a interface IMapper passa a obedecer à nova especificação da Figura 2.

<pre>public interface IMapper&lt;T&gt; {     ISqlEnumerable&lt;T&gt; GetAll();     void Update(T val);     void Delete(T val);     void Insert(T val); }</pre>	<pre>public interface ISqlEnumerable&lt;T&gt; : IEnumerable&lt;T&gt; {     ISqlEnumerable&lt;T&gt; Where(string clause); }</pre>
--	--

Figura 2

Exemplo de utilização:

```
IEnumerable<Product> prods = prodMapper  
    .GetAll()  
    .Where("CategoryID = 7")  
    .Where("UnitsInStock > 30");
```

**NOTA:** o resultado do Where deve ser avaliado de forma **Lazy** modificando em *runtime* a *query* que será executada sobre a base de dados.

## 2ª parte: associação entre EDs

Adicione suporte para a associação entre EDs, que estão relacionadas por *foreign key* na base de dados.

Uma propriedade/campo de uma ED pode ser do tipo de outra ED (associação simples 1-1), ou do tipo IEnumerable<ED> (associação múltipla 1-\*).

Por exemplo: Product pode ter uma propriedade do tipo Category (associação simples 1-1) e Supplier pode ter uma propriedade do tipo IEnumerable<Product> (associação múltipla 1-\*).

1. **[obrigatório]** suporte para associação simples.
2. **[opcional]** suporte para a associação múltipla.

Na implementação dessa funcionalidade um *data mapper* terá que recorrer ao *data mapper* da ED associada via reflexão. Para tal, torna-se útil que a interface IMapper possa ser usada sem argumentos de tipo (genéricos). Adapte a sua implementação da *framework* SqlMapper à nova definição de IMapper da Figura 3.

<pre>public interface IMapper&lt;T&gt; : IMapper {     new ISqlEnumerable&lt;T&gt; GetAll();     void Update(T val);     void Delete(T val);     void Insert(T val); }</pre>	<pre>public interface IMapper{     ISqlEnumerable GetAll();     void Update(object val);     void Delete(object val);     void Insert(object val); }</pre>
--	--

Figura 3

## 3ª Parte – Opcional: *expression trees*

Altere a implementação do método Where de ISqlEnumerable para que a cláusula seja passada na forma de uma *expression tree* em vez de uma string.

## Requisitos Transversais a todas as Partes do Trabalho

A implementação da *framework* `SqlMapper` deve obedecer aos seguintes requisitos:

- A classe que define a ED deve estar anotada com a informação do nome da tabela correspondente.
- A ED é sempre definida por um tipo referência e NÃO uma *struct*.
- Por omissão, considera-se que o nome de cada propriedade pública da ED corresponde ao nome de uma coluna da respectiva tabela.
- Na instanciação de `Builder` o programador deve poder especificar o tipo de mapeamento pretendido entre a ED e as colunas da tabela: se baseado no nome dos campos da ED; se baseado no nome das propriedades da ED; ou outro mapeamento qualquer que seja implementado à posteriori, sem necessidade de alterar o código da *framework*.
- **Pressuposto:** admite-se que o tipo do campo/propriedade da ED mapeado é compatível com o tipo do valor da coluna correspondente obtido via ADO.net, **NÃO** sendo necessário implementar nenhum suporte de conversão entre tipos.
- Os campos/propriedades da ED que corresponderem a colunas de chave primária devem estar anotados com essa informação.
- Admite-se que a *framework* só suporta tabelas com PK do tipo *identity*, NÃO sendo necessário implementar suporte para outro tipo de chaves. **OPCIONAL: suportar chaves não *identity* e compostas.**
- Na instanciação de `Builder` o programador deve poder especificar a política de gestão de ligações (`SqlConnection`), ou seja, se é reutilizada a mesma ligação em diferentes execuções dos métodos do *data mapper*, ou se é criada uma nova ligação em cada execução de cada método, ou outra estratégia qualquer de gestão de ligações que seja implementada à posteriori.
- Deve ser ainda implementada/disponibilizada uma política de gestão de ligações com suporte para iniciar e finalizar uma transacção através *rollback* ou *commit* explícito.
- O enumerável retornado por `GetAll` deve usar uma implementação **lazy**, que só faz *fetch* do `SqlDataReader` à medida que é iterado.
- Deve-se garantir que todos os recursos são *disposed* após terminada a iteração de todos os elementos, excepto numa política de *single connection* onde a ligação partilhada só deve ser fechada por ordem do programador/utilizador do `SqlMapper`.
- Como tecnologia de ligação à base de dados poderá ser usado **apenas** ADO.NET e com classes ***connected***. **NÃO podem ser usados *DataSet*, *DataTable*, nem outros meios *disconnected***.
- São valorizadas as soluções que tenham cuidados de eficiência da solução, maximizando o uso de reflexão na construção (*build*) do *data mapper* e minimizando (a zero se possível) o uso de reflexão na execução dos métodos do *data mapper*.