Instituto Superior de Engenharia de Lisboa

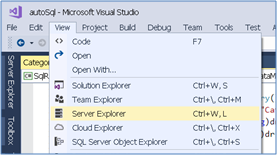
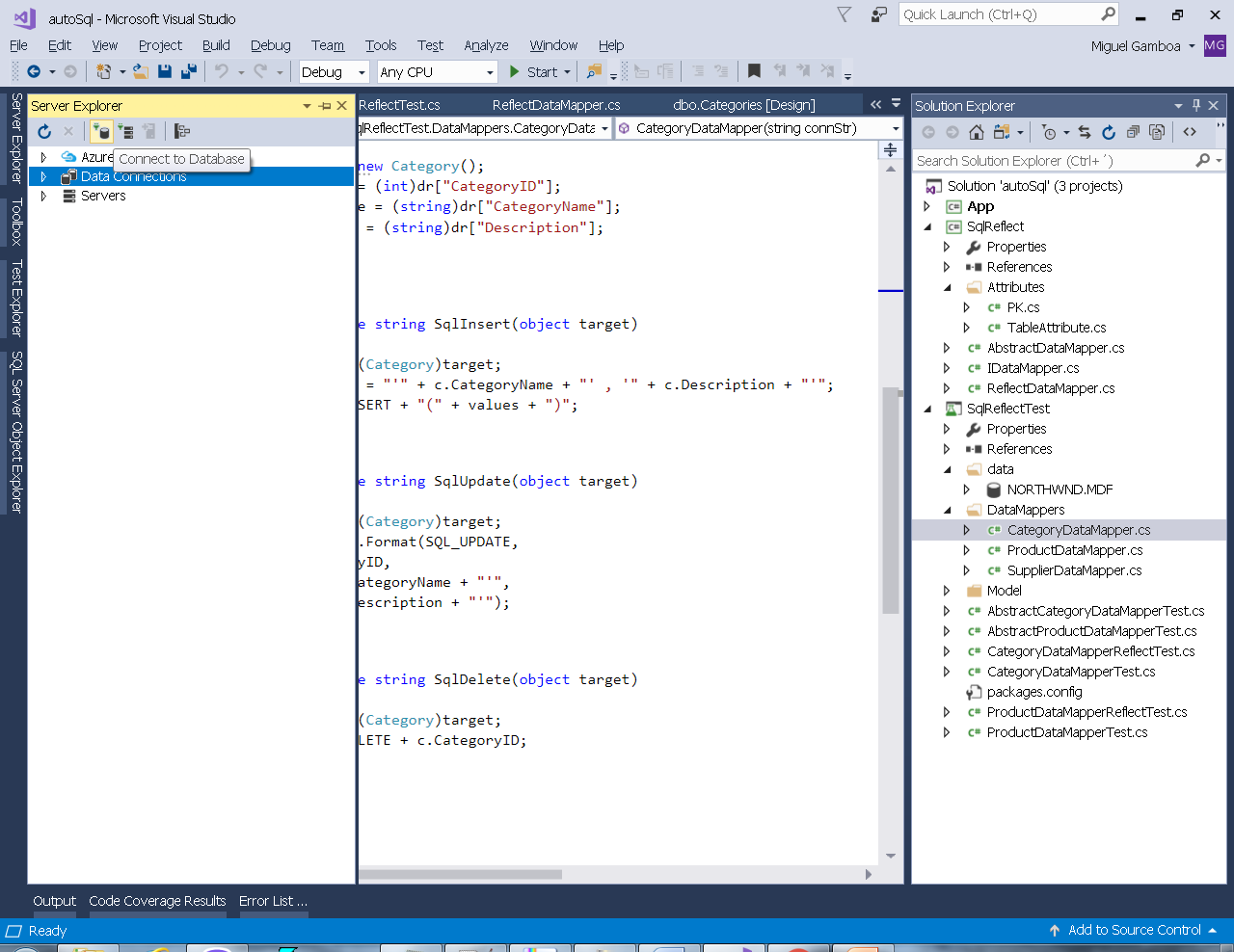
Licenciatura em Engenharia Informática e de Computadores

2018

A biblioteca **SqlReflect** permite criar instâncias de um *data mapper* (especificado pela interface IDataMapper) para qualquer tipo de entidade de domínio.

Como exemplo de utilização será usada a base de dados de SQL Server **Northwind**, que já está incluída nos projectos **App** e **SqlReflectTest** da solução autoSql.sln.

Pode visualizar o conteúdo da Northwind no VS2017 através da *view* Server Explorer e adicionando uma ligação para o ficheiro da BD SQL Server localizado em: **data\NORTHWND.MDF**.

Um [*data mapper*](https://martinfowler.com/eaaCatalog/dataMapper.html) é uma forma de organizar as operações de acesso a dados CRUD (*Create*, *Read*, *Update* e *Delete*) por entidade de domínio (e.g. Product, Employee, Region, etc.). Para cada entidade de domínio existe um *data mapper* cuja classe tem o nome **<Entidade>DataMapper** e disponibiliza os métodos correspondentes às operações CRUD.

A interface IDataMapper, presente na biblioteca **SqlReflect** tem a seguinte definição:

|  |
| --- |
| public interface IDataMapper  {  /// <summary>  /// Returns a domain object with the given id.  /// </summary>  object GetById(object id);  /// <summary>  /// Returns all rows as domain objects from the corresponding table.  /// </summary>  IEnumerable GetAll();  /// <summary>  /// Inserts the target domain object into the corresponding table.  /// </summary>  /// <returns>The identity value of the primary key column.</returns>  object Insert(object target);  /// <summary>  /// Updates the corresponding table row with the values of the target domain object.  /// </summary>  void Update(object target);  /// <summary>  /// Removes the row of the table corresponding to the target domain object.  /// </summary>  void Delete(object target);  } |

A título de exemplo o projecto **SqlReflectTest** inclui três implementações de *data* *mappers*: ProductDataMapper, CategoryDataMapper e SupplierDataMapper (este último incompleto). O objectivo da biblioteca **SqlReflect** é substituir as várias classes *data mapper* por uma única classe designada de ReflectDataMapper.

Assim podem ser obtidas novas instâncias de **IDataMapper** para qualquer entidade de domínio **sem ser necessário implementar o código** da classe *data mapper* para essa entidade. Cada instância de ReflectDataMapper representa um *data mapper* para o tipo passado no seu construtor. Exemplo:

IDataMapper categories = new ReflectDataMapper(typeof(Category), NORTHWIND)

ReflectDataMapper permite a criação de *data mappers* para entidades **sem associações** para outras entidades (e.g. Category). A tabela correspondente a uma entidade de domínio é especificada por um novo *custom attribute* de classe, TableAttribute, e a propriedade correspondente à chave primária por um novo *custom attribute* PKAttribute.

ReflectDataMapper tem ainda suporte para relações entre entidades e *data mappers*. Se uma propriedade de uma entidade é referência para outro tipo complexo (e.g. Product tem uma propriedade do tipo Category), então o seu *data mapper* depende do *data mapper* da entidade referida (e.g. ProductDataMapper depende de CategoryDataMapper).

A classe EmitDataMapper é responsável por criar uma implementação de um IDataMapper para uma determinada entidade de domínio (e.g. Supplier, Employee, etc ) usando emissão de código IL em tempo de execução.

As operações realizadas via Reflection tais como ler ou escrever propriedades de entidades de domínio passem a ser realizadas directamente com base em código IL emitido em tempo de execução através da API de System.Reflection.Emit.

De modo a suportar genéricos e iteradores *lazy*, a classe ReflectDataMapper é compatível com a interface IDataMapper<K, V>:

|  |
| --- |
| public interface IDataMapper<K, V> : IDataMapper  {  V GetById(K id);  new IEnumerable<V> GetAll();  K Insert(V target);  void Update(V target);  void Delete(V target);  } |

Os parâmetros-tipo K e V presentes na especificação de IDataMapper<K,V> designam, respectivamente, o tipo da propriedade usada como chave, e o tipo do objecto de domínio. Para tal a classe AbstractDataMapper<K,V> implementa a interface IDataMapper<K,V> e por sua vez ReflectDataMapper estende esta classe abstracta, snedo ReflectDataMapper<K,V>.

Além do suporte para genéricos a classe AbstractDataMapper<K,V> usa iteradores *lazy*.

Uma relação de *Foreign Key* pode ser implementada em OO (*Object Oriented*) como uma associação de 1:1 ou 1:N. Existe suporte para associações 1:1. Ou seja, se na tabela Products a coluna SuppierId é uma *Foreign Key* para Suppliers, então no modelo OO a entidade Product tem uma propriedade do tipo Supplier. Contudo, esta mesma relação pode ser implementada em sentido contrário com uma associação de 1:N, em que o tipo Supplier tem uma propriedade do tipo IEnumerable<Product>.

Em suma, sendo A e B entidades de domínio correspondentes às tabelas A’ e B’; se A’ tem uma FK para B’, então no modelo OO:

· A pode ter uma propriedade do tipo B – associação de 1:1

· B pode ter uma propriedade do tipo IEnumerable<A> – associação de 1:N

Para associações de 1:N existe carregamento *lazy*. Por exemplo, no modelo relacional Orders tem uma FK para Employees. Assim, no modelo OO, uma instância de Employee pode estar associada a várias ordens, tendo por isso uma propriedade IEnumerable<Order>. Neste caso o tipo Order não terá nenhuma propriedade do tipo Employee.

ReflectDataMapper<K,V> suporta propriedades deste tipo. De notar que neste exemplo uma instância de ReflectDataMapper<int, Employee> irá depender de uma instância de ReflectDataMapper<int, Order> através da qual obterá a sequência de ordens que estão associadas a um determinado empregado.

Para poder executar um comando SQL com cláusula WHERE pode tornar público o método Get(string sql) de AbstractDataMapper.