# O que é DBBroker?

É uma biblioteca .NET gratuita que tem a finalidade de tornar transparente a carga de objetos a partir de uma base de dados SQL Server através do Mapeamento de Objetos Relacionais (*Object/Relational Mapping – ORM*).

# Por que usar DBBroker?

Consultas, inserções, atualizações ou exclusões de registro na base de dados costuma ser um trabalho maçante para desenvolvedores. Isso porque geralmente exige a escrita de muitos comandos na linguagem SQL que precisam refletir os objetos, isso pode ser muito trabalhoso.

É por isso que usar o DBBroker é uma boa ideia, a maior parte do trabalho de manipulação de dados é ele quem faz. O DBBroker torna transparente toda a camada de carga de dados e as tarefas de inserção e atualização dos dados.

# Passo 1: Referenciar o DBBroker

|  |  |
| --- | --- |
| Antes de qualquer coisa é preciso fazer o download da biblioteca em [getdbroker.com](http://www.getdbbroker.com) e referenciar em seu projeto, ou instalar via NuGet no Visual Studio, se fizer assim não terá que se preocupar com a referência, pois o NuGet cuida disso para você. |  |

# Passo 2: Mapear as classes do domínio

No que diz respeito às classes de domínio, o funcionamento do DBBroker depende de 2 convenções fáceis de seguir:

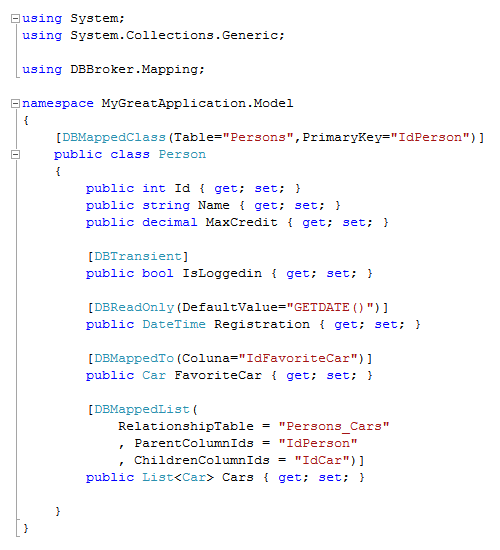
**A existência obrigatória de uma propriedade Id do tipo int;**

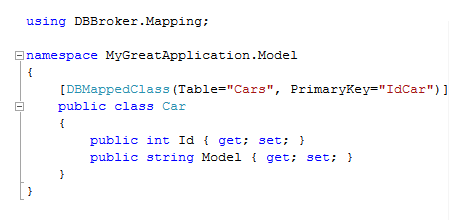
**As propriedades devem ter o mesmo nome das colunas respectivamente mapeadas (embora seja possível mapear uma propriedade para uma coluna de nome diferente, você não vai querer fazer isso, deixe esses casos serem exceção, não uma regra).**

O mapeamento das classes e propriedades se dá com anotação de 5 atributos, que são os seguintes:

* **DBMappedClass**: Mapeia as classes do domínio às tabelas e chaves primárias;
* **DBMappedList**: Usado em propriedades do tipo System.Collections.Generic.List para mapear listas;
* **DBTransient**: Para propriedades que não serão persistidas;
* **DBReadOnly**: Mapeia propriedades somente leitura, ou seja, seus valores são criados pelo banco de dados;
* **DBMappedTo**: Funciona como alias para propriedades, define o campo mapeado no banco de dados.

Nos exemplos abaixo as classes ‘Person’ e ‘Car’ utlizam todos os atributos de mapeamento.

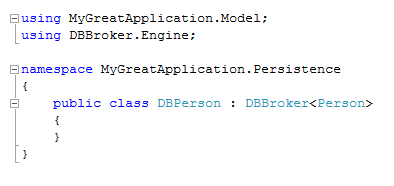




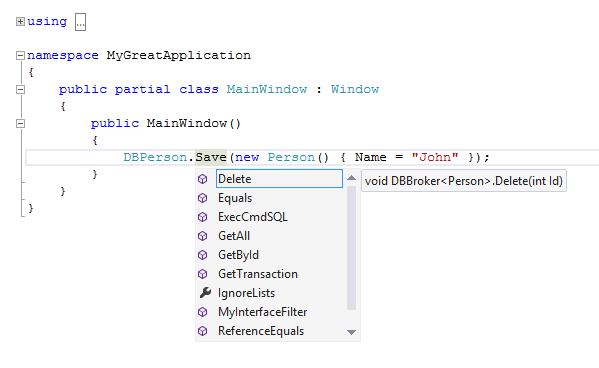
# Passo 3: Criar classes de acesso aos dados

Como convenção, para cada classe de domínio que a aplicação irá acessar diretamente, deve ser criada uma classe de acesso a dados correspondente, são nessas classes que a mágica acontece.

Essas classes devem se chamar pelo nome da classe de domínio acrescido do prefixo ‘DB’, como no exemplo abaixo.



Depois que fizer isso, você já poderá acessar os métodos: Save(), Delete(), GetById(), GetAll() e ExecCmdSQL() para manipular os dados da classe de contexto.



# Por último...

Para que o DBBroker consiga achar sua base de dados é preciso criar um arquivo de configuração incrivelmente simples no diretório base de sua aplicação, esse arquivo deve se chamar ‘DBBroker.config’.

O formato de cada linha do arquivo é composto por três valores separados pelo caractere *pipe* ‘|’: (1) Namespace do domínio mapeado, (2) o valor literal: “SQLServer” e (3) A Connection String para acesso à base SQL Server.

Ex.:

**MyGreatApplication.Model|SQLServer|Data Source=MyServer; Initial Catalog=GreatDB; User Id=minime; Password=Super§ecret**

Aproveite! Se gostou, compartilhe. Se puder, colabora ;)

Obrigado!

**Eu**quipe DBBroker