

Melhorando a performance
no .NET

Com Dapper

Por Márcio R. Nizzola



Márcio R. Nizzola
Arquiteto Sr na CI&T
Prof. na Etec de Itu

Principais tecnologias usadas
.NET
Azure

Num passado não tão distante: Angular, Ionic, Php, Visual Basic, Java,
Delphi, Pascal, C, Clipper, Basic, Cobol (entregou a idade).
marcionizzola@gmail.com



O que é o Dapper ?

É um simples mapeador de Objetos voltado à desenvolvimento com .NET.

Dapper estende a classe IDbConnection fornecendo métodos de extensão fáceis de se usar.

Teve sua primeira versão publicada em 2011 por Marc Gravell e Sam Saffron enquanto trabalhavam no Stack Overflow, para resolver problemas de performance na aplicação.

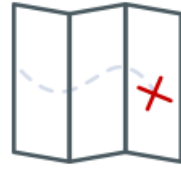
Em 2015 o site Stack Overflow registrou 5,7 bilhões de exibições de páginas usando Dapper.

Permite realizar mapeamento de consultas no banco de dados diretamente para Objetos no .NET.

Dapper não exige implementações específicas no banco de dados, ele trabalha sobre .NET ADO inclusive aceitando os bancos de dados [SQLite](#), SQL CE, Firebird, Oracle, MySQL, PostgreSQL and SQL Server.

É possível inclusive mapear para objetos não existentes nas entidades presentes no Contexto do Entity Framework da sua aplicação (exemplo: ViewModels).

Documentação: <https://dapper-tutorial.net/>



Dapper é um ORM?

Sim e não! As pessoas ainda estão discutindo sobre isso. Dapper ganhou o título de rei do C # Micro ORM, mas é considerado por várias pessoas como um mapeador de objetos simples para .NET.



O Dapper suporta Bulk Insert?

Não, mas uma biblioteca de terceiros faz: [Dapper Plus](#) . É uma biblioteca principal que estende o Dapper com todas as operações em massa.

Saber mais



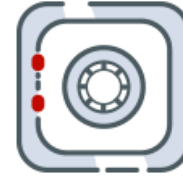
O Dapper é melhor do que o Entity Framework?

Sim e não! As pessoas preferem o Dapper quando desejam escrever a consulta SQL por conta própria com desempenho ideal.



O Dapper oferece suporte ao meu provedor de banco de dados?

Provavelmente sim, pois o Dapper fornece extensões para a interface IDbConnection. É seu trabalho escrever o SQL compatível com seu provedor de banco de dados.



O Dapper SQL Injections é seguro?

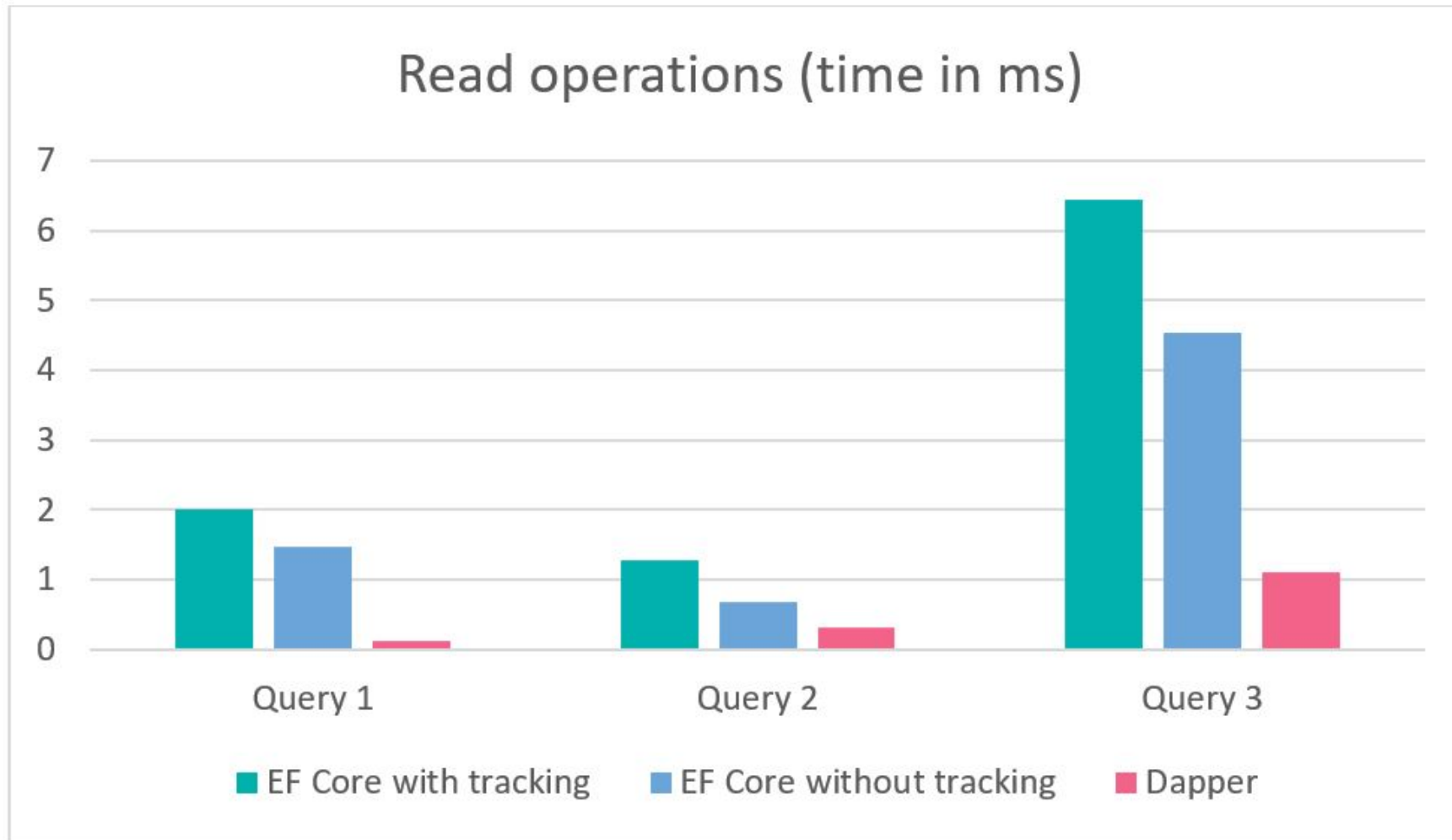
Sim, é 100% seguro se você usar consultas parametrizadas como você sempre deve fazer!



O Dapper oferece suporte a transações?

Sim, a transação de suporte Dapper e cada método que pode usar um tem um parâmetro opcional para especificá-lo.

E como é a performance ?

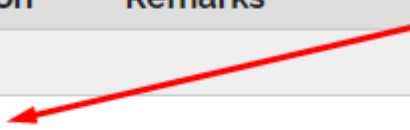


Fonte:

<https://exceptionnotfound.net/dapper-vs-entity-framework-core-query-performance-benchmarking-2019/>

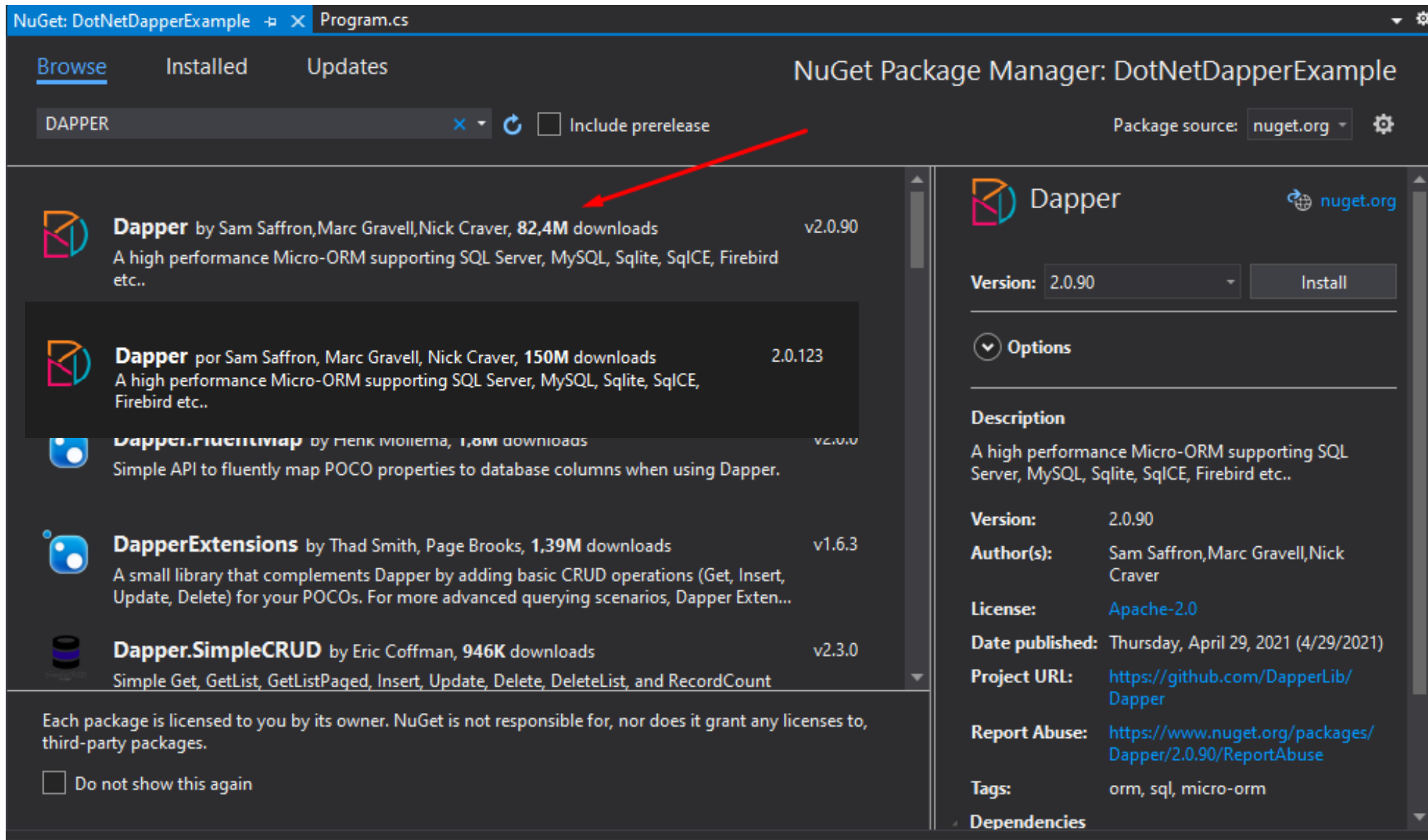
E como é a performance ?

Performance de uma pesquisa com 500 iterações - POCO serialization

Method	Duration	Remarks
Hand coded (using a SqlDataReader)	47ms	
Dapper ExecuteMapperQuery	49ms	
ServiceStack.OrmLite (QueryById)	50ms	
PetaPoco	52ms	Can be faster
BLToolkit	80ms	
SubSonic CodingHorror	107ms	
NHibernate SQL	104ms	
Linq 2 SQL ExecuteQuery	181ms	
Entity framework ExecuteStoreQuery	631ms	

Fonte: <https://www.treinaweb.com.br/blog/utilizando-o-micro-orm-dapper-em-uma-aplicacao-asp-net-core>

O Dapper até hoje (24/5/21) conta com 82 milhões de downloads !
Correção, agora em 28/06/2022 já são 150 milhões !!



The screenshot shows the NuGet Package Manager interface for a project named 'DotNetDapperExample'. The search bar contains 'DAPPER'. The results list several packages, with 'Dapper' by Sam Saffron, Marc Gravell, and Nick Craver highlighted. A red arrow points to the '82,4M downloads' text in the first result. The right-hand pane shows the details for the selected 'Dapper' package, including its version (2.0.90), description, and project URL.

NuGet Package Manager: DotNetDapperExample

Package source: [nuget.org](#)

Dapper by Sam Saffron, Marc Gravell, Nick Craver, **82,4M** downloads v2.0.90
A high performance Micro-ORM supporting SQL Server, MySQL, Sqlite, SqlCE, Firebird etc..

Dapper por Sam Saffron, Marc Gravell, Nick Craver, **150M** downloads 2.0.123
A high performance Micro-ORM supporting SQL Server, MySQL, Sqlite, SqlCE, Firebird etc..

Dapper.FluentMap by Henk Willema, **1,8M** downloads v2.0.0
Simple API to fluently map POCO properties to database columns when using Dapper.

DapperExtensions by Thad Smith, Page Brooks, **1,39M** downloads v1.6.3
A small library that complements Dapper by adding basic CRUD operations (Get, Insert, Update, Delete) for your POCOs. For more advanced querying scenarios, Dapper Exten...

Dapper.SimpleCRUD by Eric Coffman, **946K** downloads v2.3.0
Simple Get, GetList, GetListPaged, Insert, Update, Delete, DeleteList, and RecordCount

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

☐ Do not show this again

Dapper [nuget.org](#)

Version: 2.0.90 [Install](#)

Options

Description
A high performance Micro-ORM supporting SQL Server, MySQL, Sqlite, SqlCE, Firebird etc..

Version: 2.0.90
Author(s): Sam Saffron, Marc Gravell, Nick Craver
License: [Apache-2.0](#)
Date published: Thursday, April 29, 2021 (4/29/2021)
Project URL: <https://github.com/DapperLib/Dapper>
Report Abuse: <https://www.nuget.org/packages/Dapper/2.0.90/ReportAbuse>
Tags: orm, sql, micro-orm
Dependencies

Pacotes disponíveis

- o Dapper possui várias bibliotecas disponíveis, algumas de suas funcionalidades são bastante interessantes

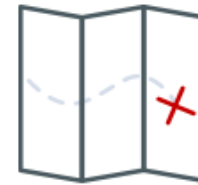
Packages				
MyGet Pre-release feed: https://www.myget.org/gallery/dapper				
Package	NuGet Stable	NuGet Pre-release	Downloads	MyGet
Dapper	nuget v2.0.90	nuget v2.0.90	downloads 82M	dapper v2.0.115
Dapper.EntityFramework	nuget v2.0.90	nuget v2.0.90	downloads 78k	dapper v2.0.115
Dapper.EntityFramework.StrongName	nuget v2.0.78	nuget v2.0.78	downloads 8.2k	dapper v2.0.115
Dapper.Rainbow	nuget v2.0.78	nuget v2.0.78	downloads 69k	dapper v2.0.115
Dapper.SqlBuilder	nuget v2.0.78	nuget v2.0.78	downloads 2.2M	dapper v2.0.115
Dapper.StrongName	nuget v2.0.90	nuget v2.0.90	downloads 2M	dapper v2.0.115

O que é o Dapper ?

Methods

Dapper will extend your `IDbConnection` interface with multiple methods:

- `Execute`
- `Query`
- `QueryFirst`
- `QueryFirstOrDefault`
- `QuerySingle`
- `QuerySingleOrDefault`
- `QueryMultiple`



Is Dapper an ORM?

Yes and no! People are still arguing about it.
Dapper has earned the title of king of the C# Micro ORM but is considered by multiple people as a simple object mapper for .NET.

Exemplo de uma consulta com ADO.NET

Prós

- Alta performance
- Facilidade para customização e conversões
- Não precisamos ficar presos à entidades de um contexto específico.

Contras

- **escrevemos muito código**
- **temos que mapear cada campo**
- **alterações de entidades obrigam revisar código**

```
public List<Aluno> GetAlunosOB()
{
    string consulta = "Select * from Alunos";
    List<Aluno> _alunos = new List<Aluno>();
    try
    {
        _connection.Open();
        SqlCommand cmd = new SqlCommand(consulta, _connection);
        cmd.CommandType = CommandType.Text;
        SqlDataReader dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            _alunos.Add(new Aluno()
            {
                Id = Convert.ToInt32(dr["Id"]),
                Nome = dr["Nome"].ToString(),
                Email = dr["Email"].ToString(),
                Curso = dr["Curso"].ToString()
            });
        }
        dr.Close();
        return _alunos;
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally
    {
        _connection.Close();
    }
}
```

Exemplo de uma consulta com Entity Framework

Prós

- escrevemos pouco código
- podemos trazer entidades e relacionamentos
- os campos são mapeados automaticamente
- alterações de campos automaticamente são resolvidas nas consultas.

```
public IEnumerable<ContaModel> GetAll()
{
    return _context.Conta.ToList();
}
```

Contras

- **performance não é um forte quando trata-se de entidades complexas**
- **trazemos apenas as entidades mapeadas no contexto**

Exemplo de uso simples do Dapper

Prós

- não precisa mapear os campos
- tem a performance similar ao ADO.NET
- escrevemos pouco código.
- Podemos trazer objetos fora do contexto.

Contras

- mudanças nos campos irão requerer alterações nas queries de consulta (depende do caso).
- dá mais trabalho que criar uma consulta no Entity Framework em entidades mais complexas.

Example usage:

```
public class Dog
{
    public int? Age { get; set; }
    public Guid Id { get; set; }
    public string Name { get; set; }
    public float? Weight { get; set; }

    public int IgnoredProperty { get { return 1; } }
}

var guid = Guid.NewGuid();
var dog = connection.Query<Dog>("select Age = @Age, Id = @Id", new { Age = (int?)null, Id = guid });

Assert.Equal(1, dog.Count());
Assert.Null(dog.First().Age);
Assert.Equal(guid, dog.First().Id);
```

tipo do objeto retornado

query

parâmetros

Exemplo de uso com múltiplos resultados

Dapper allows you to process multiple result grids in a single query.

Example:

```
var sql =
@"
select * from Customers where CustomerId = @id
select * from Orders where CustomerId = @id
select * from Returns where CustomerId = @id";

using (var multi = connection.QueryMultiple(sql, new {id=selectedId}))
{
    var customer = multi.Read<Customer>().Single();
    var orders = multi.Read<Order>().ToList();
    var returns = multi.Read<Return>().ToList();
    ...
}
```

Use of Stored Procedures

Stored Procedures

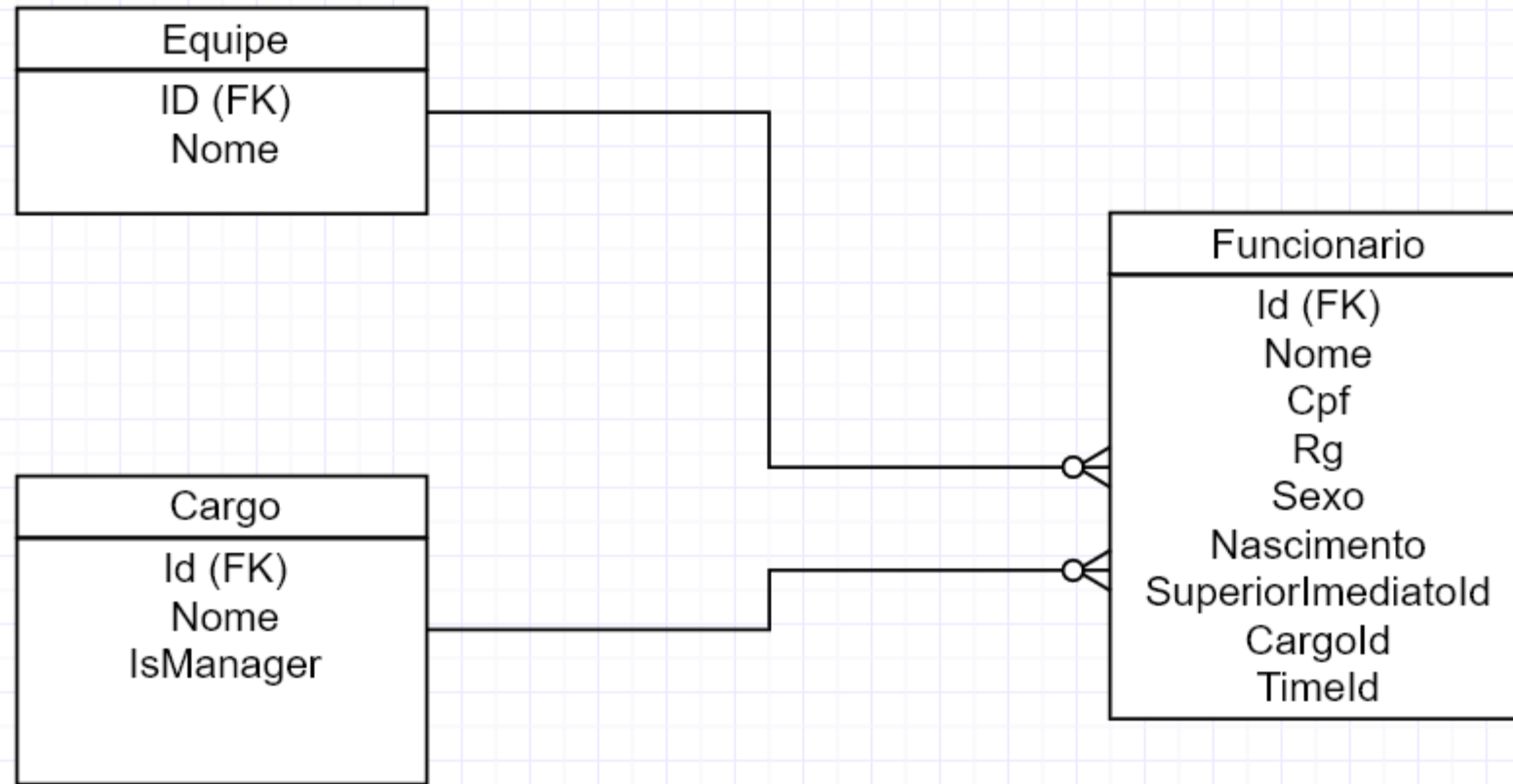
Dapper fully supports stored procs:

```
var user = cnn.Query<User>("spGetUser", new {Id = 1},  
    commandType: CommandType.StoredProcedure).SingleOrDefault();
```


Projeto Demo



Demo: Mapeamento de entidades



Demo: Mapeamento de entidades

Uso em consulta simples

```
public static string GetFuncionario = "select * from FUNCIONARIO";
```

```
0 references, 0 changes, 0 authors, 0 changes  
public ICollection<FuncionarioModel> GetFuncionarioAll()  
{  
    using (IDbConnection db = new SqlConnection(strConn))  
    {  
        var funList = db.Query<FuncionarioModel>(QueryConstants.GetFuncionario).ToList();  
        return funList;  
    }  
}
```


Demo: Mapeamento de entidades

Relacionamento de entidades de 1 para 1

Criamos na query, elementos que agirão como separadores das entidades

```
public static string GetFuncionarioCargo = @"SELECT fun.id FUN_ID,
                                           fun.*,
                                           cg.id CG_ID,
                                           CG.*

                                           FROM funcionario fun
                                           INNER JOIN cargo    cg          ON cg.id = fun.cargoid";
```

```
public ICollection<FuncionarioModel> GetFuncionarioAllCargo()
{
    using (IDbConnection db = new SqlConnection(strConn))
    {
        var funList = db.Query<FuncionarioModel, CargoModel, FuncionarioModel>(QueryConstants.GetFuncionarioCargo,
            (funcio, cargo) =>
            {
                funcio.Cargo = cargo;
                return funcio;
            }, splitOn: "FUN_ID,CG_ID").ToList();

        return funList;
    }
}
```

entidades a mapear + retorno

separadores

Demo: Mapeamento de entidades

Usamos separadores para que o Dapper entenda cada entidade separadamente.

```
public static string GetFuncionarioRelated = @"SELECT fun.id FUN_ID,
                                             fun.*,
                                             cg.id CG_ID,
                                             CG.*,
                                             TM.ID TM_ID,
                                             TM.*,
                                             SUP.ID SUP_ID,
                                             SUP.*

                                             FROM funcionario fun
                                             INNER JOIN cargo   cg      ON cg.id = fun.cargoid
                                             INNER JOIN equipe  tm      ON tm.id = fun.timeid
                                             INNER JOIN funcionario sup  ON sup.id = fun.superiorimediatoid ";
```

0 references | 0 changes | 0 authors, 0 changes

```
public ICollection<FuncionarioModel> GetFuncionarioAllRelated()
{
    using (IDbConnection db = new SqlConnection(strConn))
    {
        var funList = db.Query<FuncionarioModel, CargoModel, EquipeModel, FuncionarioModel, FuncionarioModel>(QueryConstants.GetFuncionarioRelated,
            (funcio, cargo, time, superior) =>
            {
                funcio.Cargo = cargo;
                funcio.Time = time;
                funcio.SuperiorImediato = superior;
                return funcio;
            }, splitOn: "FUN_ID,CG_ID,TM_ID,SUP_ID").ToList();

        return funList;
    }
}
```

entidades mapeadas internamente na função

entidades a mapear + entidade que retornará

separadores

Demo: Mapeamento de entidades

Tipo de retorno: Relação 1 para muitos

```
public static string GetEquipe = @"SELECT EQ.ID EQ_ID,  
                                     EQ.*,  
                                     FUN.ID FUN_ID,  
                                     FUN.*  
FROM EQUIPE EQ  
INNER JOIN FUNCIONARIO FUN ON FUN.TIMEID = EQ.ID ";
```

Cria-se uma consulta com múltiplas linhas da entidade A, para que possamos mapear o relacionamento, tudo acontecerá dentro da configuração do mapeamento.

EQ_ID	Id	Nome	FUN_ID	Id	Nome	Cpf	Rg	Sexo	Nascimento	SuperiorImediatId	TimeId	CargId
1	1	Delorean	2	2	DARTH SIDIOUS	111111111	22222222	1	1900-01-01 00:00:00.0000000	NULL	1	3
1	1	Delorean	4	4	DARTH VADER	33333441111	22222222	1	1930-01-01 00:00:00.0000000	2	1	4
1	1	Delorean	5	5	DARTH MAUL	99333441111	99222222	1	1930-01-01 00:00:00.0000000	2	1	2

Demo: Mapeamento de entidades

Tipo de retorno: Relação 1 para muitos

```
public ICollection<EquipeModel> GetEquipeAll()
{
    Dictionary<int, EquipeModel> listEquipes = new Dictionary<int, EquipeModel>();

    using (IDbConnection db = new SqlConnection(strConn))
    {
        var funList = db.Query<EquipeModel, FuncionarioModel, EquipeModel>(QueryConstants.GetEquipe,
            (equipe, funcionario) =>
            {
                if( listEquipes.TryGetValue(equipe.Id, out EquipeModel equipeItem))
                {
                    equipeItem.Integrantes.Add(funcionario);
                }
                else
                {
                    equipe.Integrantes = new List<FuncionarioModel>();
                    if( funcionario != null)
                        equipe.Integrantes.Add(funcionario);

                    listEquipes.Add(equipe.Id, equipe);
                }
                return equipe;
            }, splitOn: "EQ_ID,FUN_ID");

        return listEquipes.Values;
    }
}
```

Exemplo em:github.com/nizzola

The screenshot shows the GitHub interface for the repository **NIZZOLA / DotNetDapperExample**. The top navigation bar includes links for **Code**, **Issues**, **Pull requests**, **Actions**, **Projects**, **Wiki**, **Security**, **Insights**, and **Settings**. Below this, the repository's branch structure is shown: **main** (selected), **1 branch**, and **0 tags**. On the right, there are buttons for **Go to file**, **Add file**, and a green **Code** button with a download icon. The file list shows:

- src/DotNetDapperExample** (folder) - ajustes nos comentarios - 4 minutes ago
- .gitignore** (file) - Initial commit - 3 days ago
- readme.md** (file) - comentarios - 18 seconds ago

The **readme.md** file content is displayed below, featuring a title and a description:

readme.md

Este repositório possui exemplos do mapeamento do Dapper com C#

Foram criados exemplos de mapeamento de entidades de 1 para 1, mapeamento de entidades de 1 para muitos utilizando C#.

Minha humilde opinião

Sempre utilizei Dapper quando se trata de trazer entidades complexas com seus relacionamentos.

Já fiz inserção em lote utilizando Dapper, e foi bem performático (nos tempos do Entity Framework 5) e resolveu o problema que naquele momento exigiu uma solução.

Defendo a opinião que cada coisa deve ser utilizada para a sua finalidade, o Entity Framework faz muito bem o serviço de CRUD, portanto procuro usar dapper somente para Leituras ou caso queira uma performance Excepcional em manutenção de dados, coisa que na maioria dos sistemas não é o core da aplicação e sim leituras.

Fazer um Crud com ele dá muito mais trabalho que utilizar o Entity, assim como consultas que não precisam de muita performance (trazer uma entidade simples).

Dapper facilita a vida quando queremos trazer consultas de entidades que não estão mapeadas (Exemplo: ViewModels).

Mas cada projeto tem seus desafios, então ousem e descubram por si o melhor para cada caso, pois não existe uma bala de prata !

Referências:

<https://medium.com/geekculture/integrating-dapper-with-entity-framework-core-55aacc94b5b0>

<https://www.learndapper.com/relationships>

<https://kagawacode.medium.com/speed-benchmark-dapper-vs-ef-core-3-f3777b76dd0>

<https://medium.com/geekculture/integrating-dapper-with-entity-framework-core-55aacc94b5b0>

<https://medium.com/nerd-for-tech/using-dapper-in-asp-net-core-web-api-1d253e0e16a4>

<https://medium.com/nerd-for-tech/using-dapper-to-load-related-entities-in-net-core-e5bf4d870c7b>

<https://blog.devgenius.io/why-choose-repodb-orm-over-dapper-da87432c7830>

<https://dapper-tutorial.net/dapper-async>

FIM

