Dapper, Précis et concis

Table des matières

Repository avec Entity Framework 2
Repository avec Dapper 4
Enregistrement des repositories 6
Models avec « nullables » 6
Repository avec Stored procedures 7
Dapper contrib 9
Relations 11
Transaction avec TransactionScope 12
Plus 12

Dapper est un micro ORM, à la différence d'Entity Framework par exemple qui est un ORM (Object Relational Mapper)

Avantages :

- De meilleures performances
- On n'a pas forcément besoin d'un « framework » ORM pour une petite application
- Marche avec toutes les bases de données
- On écrit ses propres Queries

Désavantages

- SQL à écrire
- Mapping peut être difficile
- Validation supplémentaire

Dapper tutorial : https://dapper-tutorial.net/dapper

Installation du package (NuGet : https://www.nuget.org/packages/Dapper/)

```
Install-Package Dapper
```

On peut également ajouter la PackageReference directement au csproj.

Repository avec Entity Framework

On injecte le DbContext

```
namespace DapperSamples.Data.Repository
{
    public class CompanyRepositoryEF : ICompanyRepository
    {
        private readonly ApplicationDbContext _db;

        public CompanyRepositoryEF(ApplicationDbContext db)
        {
            _db = db;
        }

        public Company Add(Company company)
        {
            _db.Companies.Add(company);
            _db.SaveChanges();
            return company;
        }

        public Company Find(int id)
        {
            return _db.Companies.FirstOrDefault(u => u.CompanyId == id);
        }

        public List<Company> GetAll()
        {
            return _db.Companies.ToList();
        }

        public void Remove(int id)
```

```
        {
            Company company = _db.Companies.FirstOrDefault(u => u.CompanyId == id);
            _db.Companies.Remove(company);
            _db.SaveChanges();
            return;
        }

        public Company Update(Company company)
        {
            _db.Companies.Update(company);
            _db.SaveChanges();
            return company;
        }
    }
}
```

Exemple de DbContext (dans un projet « Data »)

```
using Microsoft.EntityFrameworkCore;

namespace DapperSamples.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {

        }

        public DbSet<Company> Companies { get; set; }
        public DbSet<Employee> Employees { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Company>().Ignore(t => t.Employees);

            modelBuilder.Entity<Employee>()
                .HasOne(c => c.Company) // 1 employee has 1 company
                .WithMany(e => e.Employees) // 1 company has many employees
                .HasForeignKey(c => c.CompanyId); // foreign key is CompanyId
        }
    }
}
```

Models

```
using System.ComponentModel.DataAnnotations;

namespace DapperSamples.Data
{
    public class Company
    {
        public Company()
        {
```

```
            Employees = new List<Employee>();
        }

        [Key]
        public int CompanyId { get; set; }
        public string? Name { get; set; }

        public string? Address { get; set; }
        public string? City { get; set; }
        public string? State { get; set; }
        public string? PostalCode { get; set; }

        public List<Employee> Employees { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace DapperSamples.Data
{
    public class Employee
    {
        [Key]
        public int EmployeeId { get; set; }
        public string? Name { get; set; }

        public string? Email { get; set; }
        public string? Phone { get; set; }
        public string? Title { get; set; }

        // foreign key
        public int CompanyId { get; set; }

        [ForeignKey("CompanyId")]
        public virtual Company? Company { get; set; }
    }
}
```

Repository avec Dapper

Exemple de repository avec Dapper

```
public interface ICompanyRepository
{
    Company Find(int id);
    List<Company> GetAll();

    Company Add(Company company);
    Company Update(Company company);

    void Remove(int id);

}
```

```
using Microsoft.Data.SqlClient;
using Microsoft.Extensions.Configuration;
using System.Data;
using Dapper;

namespace DapperSamples.Data.Repository
{
   public class CompanyRepository : ICompanyRepository
   {
      private IDbConnection db;

      public CompanyRepository(IConfiguration configuration)
      {
         this.db = new SqlConnection(configuration.GetConnectionString("DefaultConnection"));
      }

      public List<Company> GetAll()
      {
         var sql = "SELECT * FROM Companies";
         return db.Query<Company>(sql).ToList();
      }

      public Company Find(int id)
      {
         var sql = "SELECT * FROM Companies WHERE CompanyId = @CompanyId";
         return db.Query<Company>(sql, new { @CompanyId = id }).Single();
      }

      public Company Add(Company company)
      {
         var sql = "INSERT INTO Companies (Name, Address, City, State, PostalCode) VALUES(@Nam
@Address, @City, @State, @PostalCode);"
                  + "SELECT CAST(SCOPE_IDENTITY() as int); ";
         var id = db.Query<int>(sql, company).Single();
         company.CompanyId = id;
         return company;

      }

      public void Remove(int id)
      {
         var sql = "DELETE FROM Companies WHERE CompanyId = @Id";
         db.Execute(sql, new { id });
      }

      public Company Update(Company company)
      {
         var sql = "UPDATE Companies SET Name = @Name, Address = @Address, City = @City, " +
            "State = @State, PostalCode = @PostalCode WHERE CompanyId = @CompanyId";
         db.Execute(sql, company);
         return company;
      }
   }
}
```

On récupère l'Id que l'on affecte à entity

Il est possible de créer des versions Async des méthodes. Exemple

```
public async Task<Employee> AddAsync(Employee employee)
{
   var sql = "INSERT INTO Employees (Name, Title, Email, Phone, CompanyId) VALUES(@Name, @Title,
@Email, @Phone, @CompanyId);"
             + "SELECT CAST(SCOPE_IDENTITY() as int); ";
   var id = await db.QueryAsync<int>(sql, employee);
   employee.EmployeeId = id.Single();
   return employee;
}
```

Enregistrement des repositories

Exemple avec une application Asp.Net Core Mvc .NET 6

```
var builder = WebApplication.CreateBuilder(args);

// builder.Services.AddScoped<ICompanyRepository, CompanyRepositoryEF>();
//builder.Services.AddScoped<ICompanyRepository, CompanyRepository>();
builder.Services.AddScoped<ICompanyRepository, CompanyRepositorySP>();
builder.Services.AddScoped<IEmployeeRepository, EmployeeRepository>();
```

Que l'on injecte dans les controllers

Models avec « nullables »

Il faut mettre en nullable avec « ? » toutes les properties (**pas seulement les types primitifs**) pouvant ne pas avoir de valeur. C'est d'autant plus vrai avec Entity Framework pour la génération des migrations et de la base de données.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace DapperSamples.Data
{
   public class Employee
   {
      [Key]
      public int EmployeeId { get; set; }
      public string? Name { get; set; }

      public string? Email { get; set; }
      public string? Phone { get; set; }
      public string? Title { get; set; }

      // foreign key
      public int CompanyId { get; set; }

      [ForeignKey("CompanyId")]
      public virtual Company? Company { get; set; }
   }
}
```

Repository avec Stored procedures

```
using Dapper;
using Microsoft.Data.SqlClient;
using Microsoft.Extensions.Configuration;
using System.Data;

namespace DapperSamples.Data.Repository
{
   public class CompanyRepositorySP : ICompanyRepository
   {
      private IDbConnection db;

      public CompanyRepositorySP(IConfiguration configuration)
      {
         this.db = new SqlConnection(configuration.GetConnectionString("DefaultConnection"));
      }

      public List<Company> GetAll()
      {
         return db.Query<Company>("usp_GetALLCompany", commandType:
CommandType.StoredProcedure).ToList();
      }

      public Company Find(int id)
      {
         return db.Query<Company>("usp_GetCompany", new { CompanyId = id }, commandType:
CommandType.StoredProcedure).SingleOrDefault();
      }

      public Company Add(Company company)
      {
         var parameters = new DynamicParameters();
         parameters.Add("@CompanyId", 0, DbType.Int32, direction: ParameterDirection.Output);
         parameters.Add("@Name", company.Name);
         parameters.Add("@Address", company.Address);
         parameters.Add("@City", company.City);
         parameters.Add("@State", company.State);
         parameters.Add("@PostalCode", company.PostalCode);

         this.db.Execute("usp_AddCompany", parameters, commandType: CommandType.StoredProcedure);
         company.CompanyId = parameters.Get<int>("CompanyId");

         return company;
      }

      public Company Update(Company company)
      {
         var parameters = new DynamicParameters();
         parameters.Add("@CompanyId", company.CompanyId, DbType.Int32);
         parameters.Add("@Name", company.Name);
         parameters.Add("@Address", company.Address);
         parameters.Add("@City", company.City);
         parameters.Add("@State", company.State);
         parameters.Add("@PostalCode", company.PostalCode);
         this.db.Execute("usp_UpdateCompany", parameters, commandType: CommandType.StoredProcedure);

         return company;
```

```
        }

        public void Remove(int id)
        {
            db.Execute("usp_RemoveCompany", new { CompanyId = id }, commandType:
CommandType.StoredProcedure);
        }
    }
}
```

Les procédures crées (migration avec Entity Framework)

```
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace DapperSamples.Data.Migrations
{
    public partial class AddStoredProcedures : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(@"
                CREATE PROC usp_GetCompany
                    @CompanyId int
                AS
                BEGIN
                    SELECT *
                    FROM Companies
                    WHERE CompanyId = @CompanyId
                END
                GO
            ");

            migrationBuilder.Sql(@"
                CREATE PROC usp_GetALLCompany
                AS
                BEGIN
                    SELECT *
                    FROM Companies
                END
                GO
            ");

            migrationBuilder.Sql(@"
                CREATE PROC usp_AddCompany
                    @CompanyId int OUTPUT,
                    @Name varchar(MAX),
                            @Address  varchar(MAX),
                            @City varchar(MAX),
                            @State varchar(MAX),
                            @PostalCode varchar(MAX)
                AS
                BEGIN
                    INSERT INTO Companies (Name, Address, City, State, PostalCode) VALUES(@Name, @Address, @City, @State,
@PostalCode);
                            SELECT @CompanyId = SCOPE_IDENTITY();
                END
                GO
            ");

            migrationBuilder.Sql(@"
                CREATE PROC usp_UpdateCompany
```

```
                @CompanyId int,
            @Name varchar(MAX),
                @Address  varchar(MAX),
                @City varchar(MAX),
                @State varchar(MAX),
                @PostalCode varchar(MAX)
        AS
        BEGIN
            UPDATE Companies
                SET
                    Name = @Name,
                    Address = @Address,
                    City=@City,
                    State=@State,
                    PostalCode=@PostalCode
                WHERE CompanyId=@CompanyId;
                SELECT @CompanyId = SCOPE_IDENTITY();
        END
        GO
    ");

    migrationBuilder.Sql(@"
        CREATE PROC usp_RemoveCompany
            @CompanyId int
        AS
        BEGIN
            DELETE
            FROM Companies
            WHERE CompanyId  = @CompanyId
        END
        GO
    ");
  }

  protected override void Down(MigrationBuilder migrationBuilder)
  {

  }
 }
}
```

Dapper contrib

https://github.com/DapperLib/Dapper.Contrib

NuGet : https://www.nuget.org/packages/Dapper.Contrib/

```
Install-Package Dapper.Contrib
```

Plus besoin de faire des queries avec le SQL

Repository avec Dapper Contrib

```
using Microsoft.Data.SqlClient;
using Microsoft.Extensions.Configuration;
using System.Data;
using Dapper.Contrib.Extensions;

namespace DapperSamples.Data.Repository
```

```
{
    public class CompanyRepositoryContib : ICompanyRepository
    {
        private IDbConnection db;

        public CompanyRepositoryContib(IConfiguration configuration)
        {
            this.db = new SqlConnection(configuration.GetConnectionString("DefaultConnection"));
        }

        public List<Company> GetAll()
        {
            return db.GetAll<Company>().ToList();
        }

        public Company Find(int id)
        {
            return db.Get<Company>(id);
        }

        public Company Add(Company company)
        {
            var id = db.Insert(company);
            company.CompanyId = (int)id;
            return company;
        }

        public Company Update(Company company)
        {
            db.Update(company);
            return company;
        }

        public void Remove(int id)
        {
            db.Delete(new Company { CompanyId = id });
        }
    }
}
```

Note il faut indiquer la « Key » (auto incrément) ou « ExplicitKey » (attention de ne pas confondre l'attribut des Data Annotations)

```
using Dapper.Contrib.Extensions;

namespace DapperSamples.Data
{
    [Table("Companies")]
    public class Company
    {
        public Company()
        {
            Employees = new List<Employee>();
        }
```

Permet d'éviter d'avoir une exception de « nom » sur GetAll qui chercherait « companys »

```
        [Key]
        public int CompanyId { get; set; }

        public string? Name { get; set; }

        public string? Address { get; set; }
        public string? City { get; set; }
        public string? State { get; set; }
        public string? PostalCode { get; set; }

        [Write(false)]
        public List<Employee> Employees { get; set; }
    }
}
```

Relations

1-1 Obtenir la company de chaque employee

```
public List<Employee> GetEmployeeWithCompany()
{
    var sql = "SELECT E.*,C.* FROM Employees AS E INNER JOIN Companies AS C ON E.CompanyId =
C.CompanyId ";
    var employee = db.Query<Employee, Company, Employee>(sql, (e, c) =>
    {
        e.Company = c;
        return e;
    }, splitOn: "CompanyId");

    return employee.ToList();
}
```

1-N Obtenir tous les employees d'une company pour afficher le detail

```
public class CompanyRepository : ICompanyRepository
{
    private IDbConnection db;

    public CompanyRepository(IConfiguration configuration)
    {
        this.db = new SqlConnection(configuration.GetConnectionString("DefaultConnection"));
    }

    public Company GetCompanyWithEmployees(int id)
    {
        var p = new
        {
            CompanyId = id
        };

        var sql = "SELECT * FROM Companies WHERE CompanyId = @CompanyId;"
```

```
        + " SELECT * FROM Employees WHERE CompanyId = @CompanyId; ";

      Company company;

      using (var lists = db.QueryMultiple(sql, p))
      {
         company = lists.Read<Company>().ToList().FirstOrDefault();
         company.Employees = lists.Read<Employee>().ToList();
      }

      return company;
   }
}
```

Transaction avec TransactionScope

Exemple

```
public void TestTransaction(Company objComp)
{
    using (var transaction = new TransactionScope())
    {
        try
        {
            var sql = "INSERT INTO Companies (Name, Address, City, State, PostalCode) VALUES(@Name,
@Address, @City, @State, @PostalCode);"
            + "SELECT CAST(SCOPE_IDENTITY() as int); ";
            var id = db.Query<int>(sql, objComp).Single();
            objComp.CompanyId = id;

            objComp.Employees.Select(c => { c.CompanyId = id; return c; }).ToList();
            var sqlEmp = "INSERT INTO Employees (Name, Title, Email, Phone, CompanyId) VALUES(@Name,
@Title, @Email, @Phone, @CompanyId);"
                        + "SELECT CAST(SCOPE_IDENTITY() as int); ";
            db.Execute(sqlEmp, objComp.Employees);

            transaction.Complete();
        }
        catch(Exception ex)
        { }
    }
}
```

Plus

Remove range « in »

```
public void RemoveRange(int[] companyId)
{
    db.Query("DELETE FROM Companies WHERE CompanyId IN @companyId", new {
companyId });
}
```

Filter « like »

```
public List<Company> FilterCompanyByName(string name)
{
    return db.Query<Company>("SELECT * FROM Companies WHERE Name like '%' +
@name + '%' ", new { name }).ToList();
```

```
}
```

Bulk

https://dapper-tutorial.net/bulk-insert