The Learning House

427 S 4th Street #300

Louisville KY 40202

Intro To Dapper.NET

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

Learn about the Dapper.NET extensions for ADO.NET

What is Dapper.NET

Dapper is an *object-relational mapping* (ORM) solution that provides quick and easy methods for binding SQL query data to .NET objects.

Dapper is free and open source. It was created by StackOverflow's development team in response to performance issues with Entity Framework.

Dapper is a Micro-ORM

This means it does not offer the full range of features as in a full ORM like Entity Framework or nHibernate.

This is by design. It focuses on simplicity and performance rather than full automation, which effectively means it doesn't generate its own SQL queries, but maps results to C# objects.

One reason it is popular is because it plays nicely with stored procedures.

Getting Started

We have to make sure to add a connection string to our config file and create a class to read in the connection string.

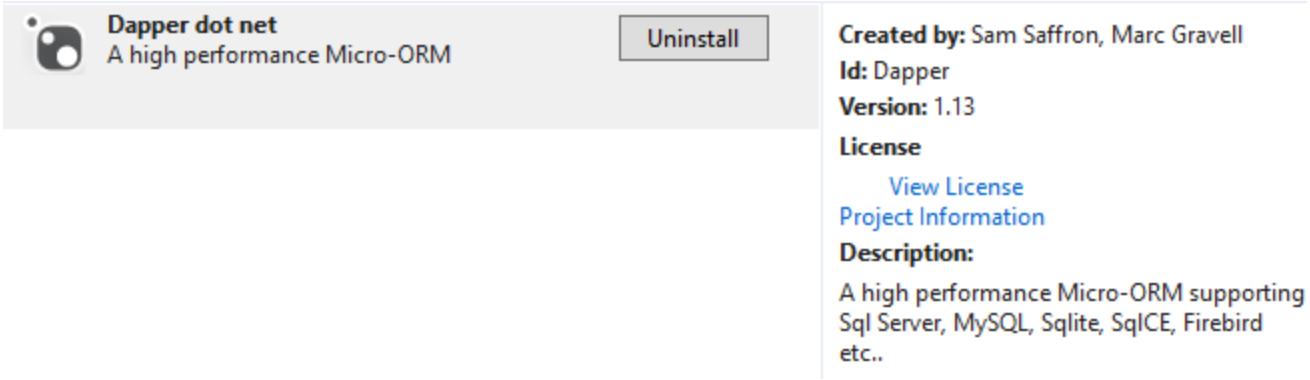
```
public static class Settings
{
    private static string _connectionString;

    public static string ConnectionString
    {
        get
        {
            if (string.IsNullOrEmpty(_connectionString))
            {
                _connectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
            }

            return _connectionString;
        }
    }
}
```

We Also Need to Get Dapper.Net

NuGet, of course!



Running a SQL Query

We still want to create a connection with a using statement for clean-up, but notice we no longer need a command object:

```
using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
{
    var employees = cn.Query<Employee>("SELECT * FROM Employees").ToList();

    Assert.AreEqual(9, employees.Count);
}
```

Dapper adds a Query<T> method to the connection class, which returns an IEnumerable. If we want a list, we can just call .ToList().

Running a Query with Parameters

Here we have a choice: we can declare parameters in-line for the Query method or declare a DynamicParameters object and pre-define them.

```
        using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
        {
            var employees = cn.Query<Employee>("SELECT * FROM Employees WHERE EmployeeId > @EmployeeId",
                new{EmployeeId=5}).ToList();

            Assert.AreEqual(4, employees.Count);
        }
```

```
using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
{
    var p = new DynamicParameters();
    p.Add("EmployeeId", 5);
    var employees = cn.Query<Employee>("SELECT * FROM Employees WHERE EmployeeId > @EmployeeId", p).ToList();

    Assert.AreEqual(4, employees.Count);
}
```

Getting One Result

Query<T>() always returns IEnumerable, so if we only want one object, we need to use FirstOrDefault().

```
using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
{
    var p = new DynamicParameters();
    p.Add("EmployeeId", 5);
    var employee = cn.Query<Employee>("SELECT * FROM Employees WHERE EmployeeId = @EmployeeId", p).FirstOrDefault();

    Assert.IsNotNull(employee);
}
```

```
using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
{
    var p = new DynamicParameters();
    p.Add("EmployeeId", 25);
    var employee = cn.Query<Employee>("SELECT * FROM Employees WHERE EmployeeId = @EmployeeId", p).FirstOrDefault();

    Assert.IsNull(employee);
}
```

Executing Stored Procedures

This uses pretty much the same code as running in-line queries, except you have to tell the Query method that the command is a stored procedure:

```
using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
{
    var employees = cn.Query<Employee>("EmployeeGetAll",
        commandType: CommandType.StoredProcedure).ToList();

    Assert.AreEqual(9, employees.Count);
}
```

Non Query / Returning Output Parameters

This is a common pattern. Say we want to insert a record and retrieve the primary key created:

```
CREATE PROCEDURE RegionInsert (
    @RegionDescription nchar(50),
    @RegionId int output
) AS
BEGIN

    INSERT INTO Region (RegionDescription)
    VALUES (@RegionDescription);

    SET @RegionId = SCOPE_IDENTITY();

END
```

C# Code

1. Add an output parameter to the DynamicParameters object
2. Execute the command
3. Get the value of the parameter.

```
using (SqlConnection cn = new SqlConnection(Settings.ConnectionString))
{
    var p = new DynamicParameters();
    p.Add("RegionDescription", "New Region");
    p.Add("RegionId", DbType.Int32, direction: ParameterDirection.Output);

    cn.Execute("RegionInsert", p, commandType: CommandType.StoredProcedure);

    int regionId = p.Get<int>("RegionId");

    Assert.AreNotEqual(0, regionId);
}
```

Use Execute Instead of Query…

… when you are not returning a result set from SQL (Insert, Update, Delete).