Microsoft

Dapper

Crystal Tenn
Crystal.Tenn@microsoft.com

ORM

- ORM (Object Relational Mapper) is a tool that creates layer between your application and data source and returns you the relational objects instead of (in terms of C# that you are using) ADO.NET objects. This is basic thing that every ORM does.

- EF is considered a full ORM framework
- Dapper is a micro ORM

Let's start by looking at performance metrics for ADO.NET vs Dapper vs EF...

Performance

- https://www.exceptionnotfound.net/dapper-vs-entity-framework-vs-ado-net-performance-benchmarking/

- The following results are for 10 iterations, each containing 8 sports, 30 teams in each sport, and 100 players per team.

ADO.NET vs EF

ADO.NET Results

Run	Player by ID	Players for Team	Teams for Sport
1	0.01ms	1.03ms	10.25ms
2	0ms	1ms	11ms
3	0.1ms	1.03ms	9.5ms
4	0ms	1ms	9.62ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1ms	8.12ms
9	0ms	1ms	8ms
10	0ms	1.17ms	8.88ms
Average	**0.013ms**	**1.03ms**	**8.84ms**

Entity Framework Results

Run	Player by ID	Players for Team	Teams for Sport
1	1.64ms	4.57ms	127.75ms
2	0.56ms	3.47ms	112.5ms
3	0.17ms	3.27ms	119.12ms
4	1.01ms	3.27ms	106.75ms
5	1.15ms	3.47ms	107.25ms
6	1.14ms	3.27ms	117.25ms
7	0.67ms	3.27ms	107.25ms
8	0.55ms	3.27ms	110.62ms
9	0.37ms	4.4ms	109.62ms
10	0.44ms	3.43ms	116.25ms
Average	**0.77ms**	**3.57ms**	**113.45ms**

ADO.NET vs Dapper

ADO.NET Results

Run	Player by ID	Players for Team	Teams for Sport
1	0.01ms	1.03ms	10.25ms
2	0ms	1ms	11ms
3	0.1ms	1.03ms	9.5ms
4	0ms	1ms	9.62ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1ms	8.12ms
9	0ms	1ms	8ms
10	0ms	1.17ms	8.88ms
Average	**0.013ms**	**1.03ms**	**8.84ms**

Dapper.NET Results

Run	Player by ID	Players for Team	Teams for Sport
1	0.38ms	1.03ms	9.12ms
2	0.03ms	1ms	8ms
3	0.02ms	1ms	7.88ms
4	0ms	1ms	8.12ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1.02ms	7.62ms
9	0ms	1ms	7.88ms
10	0.02ms	1ms	7.75ms
Average	**0.047ms**	**1.01ms**	**7.94ms**

EF vs Dapper

Entity Framework Results

Run	Player by ID	Players for Team	Teams for Sport
1	1.64ms	4.57ms	127.75ms
2	0.56ms	3.47ms	112.5ms
3	0.17ms	3.27ms	119.12ms
4	1.01ms	3.27ms	106.75ms
5	1.15ms	3.47ms	107.25ms
6	1.14ms	3.27ms	117.25ms
7	0.67ms	3.27ms	107.25ms
8	0.55ms	3.27ms	110.62ms
9	0.37ms	4.4ms	109.62ms
10	0.44ms	3.43ms	116.25ms
Average	**0.77ms**	**3.57ms**	**113.45ms**

Dapper.NET Results

Run	Player by ID	Players for Team	Teams for Sport
1	0.38ms	1.03ms	9.12ms
2	0.03ms	1ms	8ms
3	0.02ms	1ms	7.88ms
4	0ms	1ms	8.12ms
5	0ms	1.07ms	7.62ms
6	0.02ms	1ms	7.75ms
7	0ms	1ms	7.62ms
8	0ms	1.02ms	7.62ms
9	0ms	1ms	7.88ms
10	0.02ms	1ms	7.75ms
Average	**0.047ms**	**1.01ms**	**7.94ms**

EF vs Dapper

Performance of SELECT mapping over 500 iterations - 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	

https://github.com/StackExchange/Dapper

EF vs Dapper

There's one other point to be made about comparing "apples to apples." Dapper takes in raw SQL. By default, EF queries are expressed with LINQ to EF and must go through some effort to build the SQL for you. Once that SQL is built, even SQL that relies on parameters, it's cached in the application's memory so that effort is reduced on repetition. Additionally, EF has the ability to execute queries using raw SQL, so I've taken both approaches into account. **Figure 3** lists the comparative results of four sets of tests. The download contains even more tests.

Figure 3 Average Time in Milliseconds to Execute a Query and Populate an Object Based on 25 Iterations, Eliminating the Fastest and Slowest

*AsNoTracking queries	Relationship	LINQ to EF*	EF Raw SQL*	Dapper Raw SQL
All Designers (30K rows)	–	96	98	77
All Designers with Products (30K rows)	1 : *	251	107	91
All Designers with Clients (30K rows)	* : *	255	106	63
All Designers with Contact (30K rows)	1 : 1	322	122	116

Reference: https://msdn.microsoft.com/en-us/magazine/mt703432.aspx

Raw SQL queries using EF: https://msdn.microsoft.com/en-us/library/jj592907(v=vs.113).aspx

What is Dapper?

Dapper

- Dapper is a simple object mapper for .NET and owns the title of King of Micro ORM in terms of speed and is virtually as fast as using a raw ADO.NET data reader. An ORM is an Object Relational Mapper, which is responsible for mapping between database and programming language.

- Dapper extends the IDbConnection by providing useful extension methods to query your database.

Why Dapper?

Dapper

- Dapper has an interesting history, having spawned from a resource you might be extremely familiar with: Marc Gravell and Sam Saffron built Dapper while working at Stack Overflow, solving performance issues with the platform.

- In 2011, Saffron wrote a blog post about the work he and Gravell had done, titled, “How I Learned to Stop Worrying and Write My Own ORM” (aka.ms/Vqpql6), which explains the performance issues Stack was having at the time, stemming from its use of LINQ to SQL.

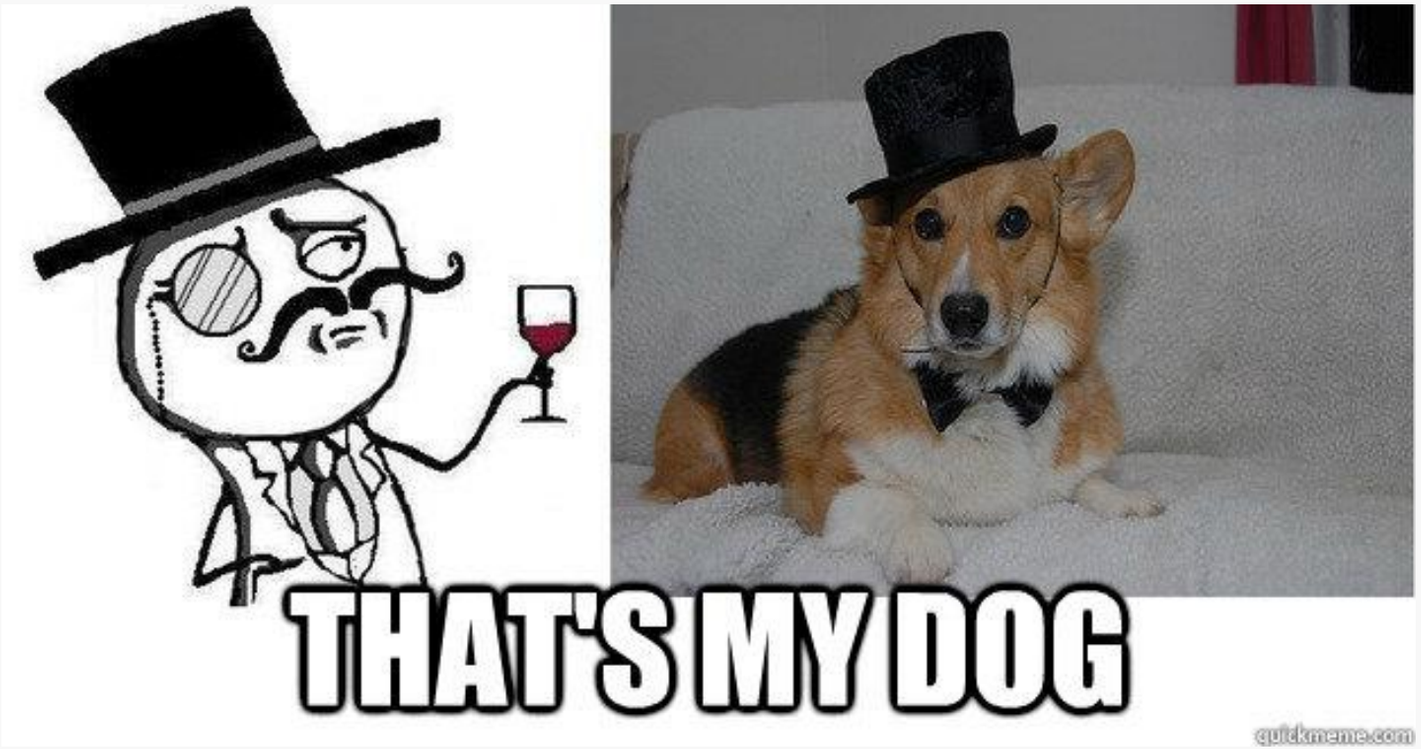
- He then details why writing a custom ORM, Dapper, was the answer for optimizing data access on Stack Overflow. Five years later, Dapper is now widely used and open source. Gravell and Stack and team member Nick Craver continue to actively manage the project at github.com/StackExchange/dapper-dot-net.

https://msdn.microsoft.com/en-us/magazine/mt703432.aspx

What is Dapper?

3 step process to Dapper

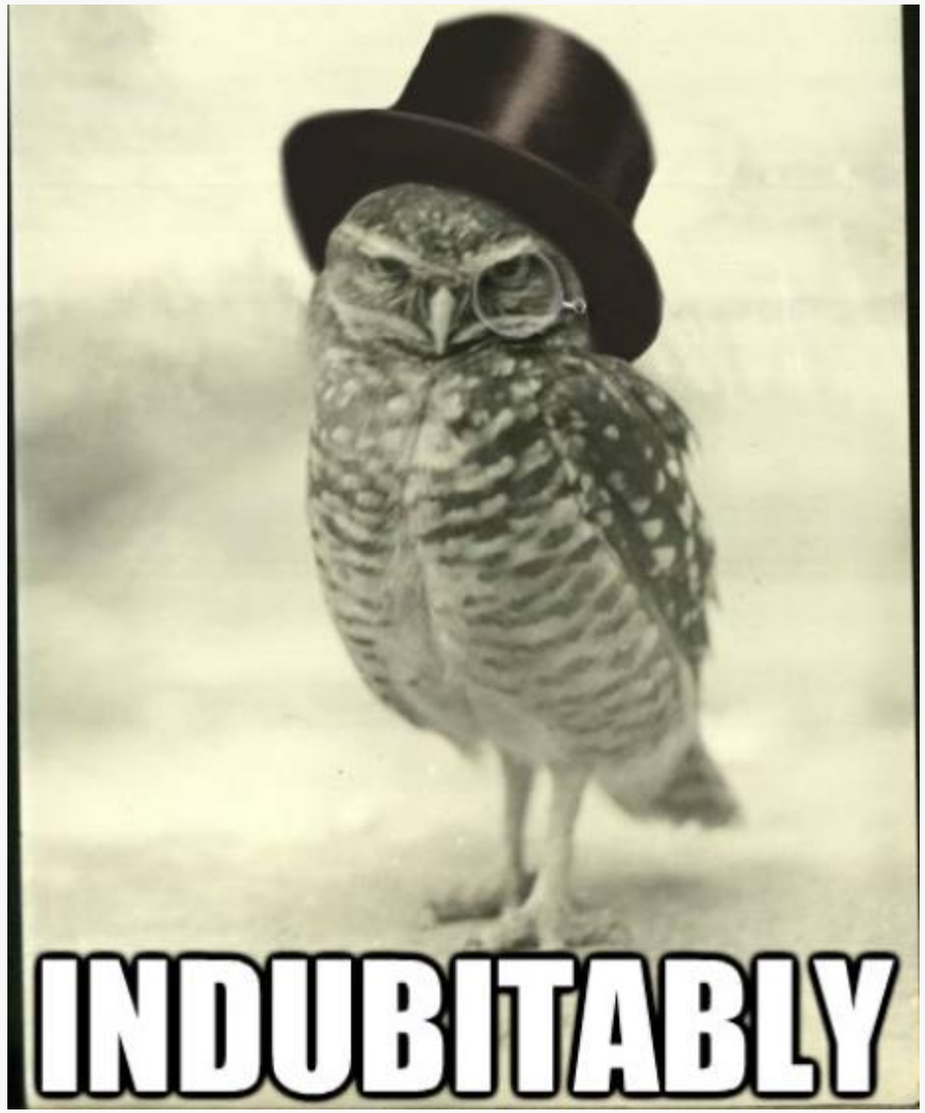
1. Create an IDbConnection object.
2. Write a query to perform CRUD operations.
3. Pass query as a parameter in Execute method.



3 step process to Dapper

- Dapper works with any database provider since there is no DB specific implementation.
- Dapper will extend your IDbConnection interface with multiple methods:

- Execute
- Query *get me a list of objects
- QueryFirst
- QueryFirstOrDefault
- QuerySingle *get me the first result
- QuerySingleOrDefault
- QueryMultiple *more than 1 select statement



Query

```
string sql = "SELECT * FROM Invoice;";

using (var connection = My.ConnectionFactory())
{
    connection.Open();

    var invoices = connection.Query(sql).ToList();

    My.Result.Show(invoices);
}
```

Query

```
string sql = "EXEC Invoice_Insert";

using (var connection = My.ConnectionFactory())
{
    connection.Open();

    var affectedRows = connection.Execute(sql,
        new {Kind = InvoiceKind.WebInvoice, Code = "Single_Insert_1"},
        commandType: CommandType.StoredProcedure);

    My.Result.Show(affectedRows);
}
```

Query Mutliple

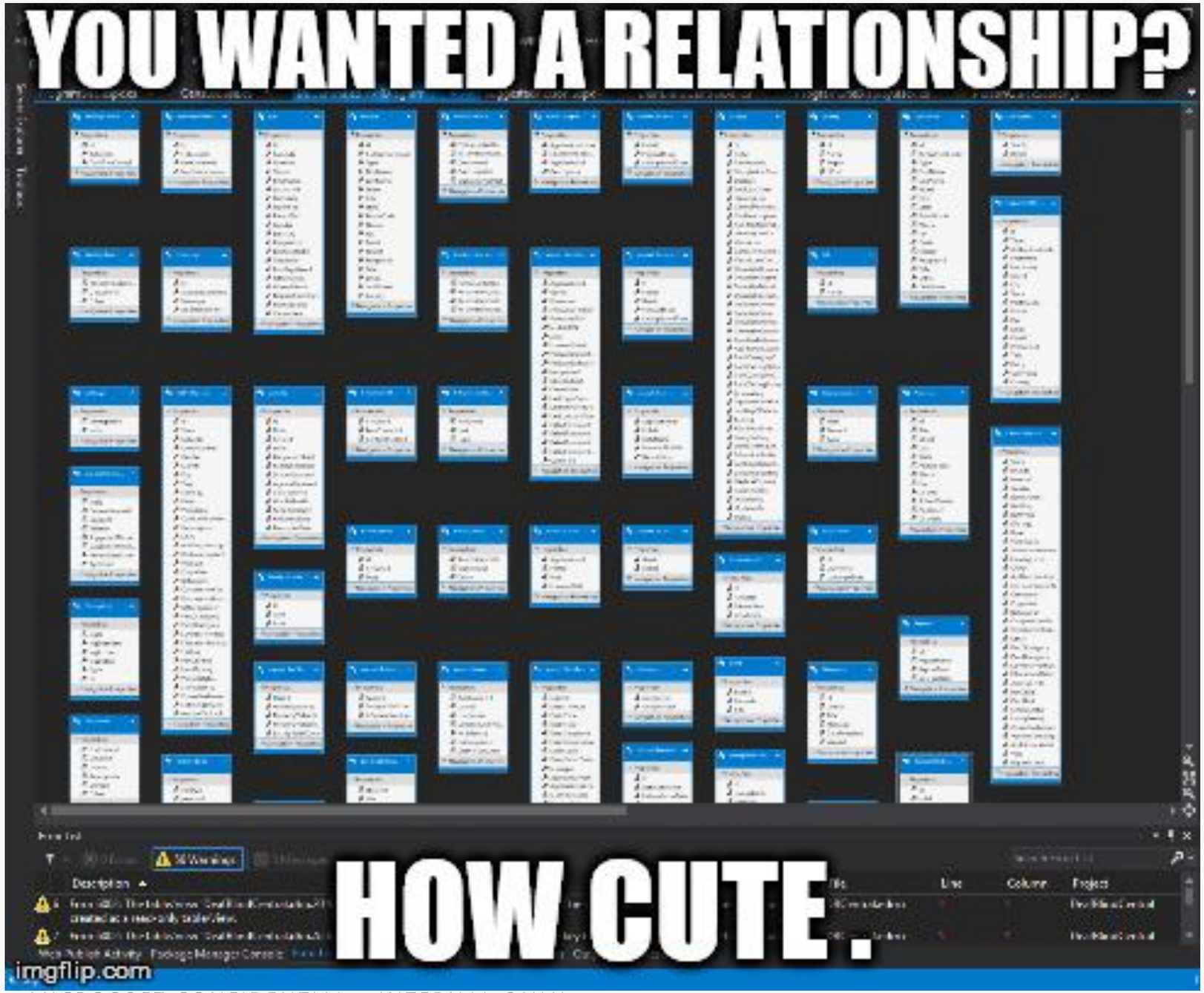
- Notice more than 1 select statement
- 2 Selects here will get you 2 result sets, the Grid Reader is used to read them
- “var multi” is a Grid Reader
- It is a way of getting multiple result sets, and you have to use the .Read method to get each result set out.

```
string sql = "SELECT * FROM Invoice WHERE InvoiceID = @InvoiceID; SELECT * FROM InvoiceItem WHERE InvoiceID = @InvoiceID;";

using (var connection = My.ConnectionFactory())
{
    connection.Open();

    using (var multi = connection.QueryMultiple(sql, new {InvoiceID = 1}))
    {
        var invoice = multi.Read<Invoice>().First();
        var invoiceItems = multi.Read<InvoiceItem>().ToList();
    }
}
```

What is Entity Framework?

Entity Framework

- **Entity Framework** (EF) is an open source object-relational mapping (ORM) **framework** for ADO.NET. It was a part of .NET **Framework**, but since **Entity framework** versio n 6 it is separated from .NET **framework**.



Entity Framework

Features for Database First (Designer)

and Code-First Both:

- Connection resiliency
- Asynchronous query and save
- Code-based configuration
- Database command logging
- Database command interception
- Dependency Resolution
- DbSet.AddRange/RemoveRange
- Better Transaction Support
- Pluggable pluralisation and singularization service
- Testability improvements
- Creating context with an open connection
- Improved performance and warm-up time

Features for Code-First:

- Custom conventions
- Insert, update & delete stored procedures for entity CUD operation
- Index attribute (EF 6.1)
- Multiple context per database
- Nested entity types
- Custom migration operations
- Configurable migration history table

Lab on Dapper

- 01 ToDoListDataAPI [begin], continued from your work in Lab 1 on Swagger. We will convert the ADO.NET in the application to Dapper.

Microsoft

Thank you!