Team “Hello World” 12/02/2020
Team Lead: Matthew Caponi

Matthew Caponi - 011161928
Shaan Shariff - 015741750
Wonsuk Seo - 018660874
Ian Macabulos - 016401240
Jakub Koziol - 016523856

Object-Relational Mapper Approval Request

Business Need

An object-relational mapper is a tool that acts as an interface between a database server and an IDE. It helps to convert database objects into objects that can be understood by OOP IDE’s. This is essential to our application, as any good data access layer needs to be capable of communicating with the database in a simple manner. Since our application will need to maintain multiple databases, having an ORM will be essential to build and maintain our data access layer.

The Options

Dapper

Dapper is a lightweight, micro-ORM built specifically for .NET. A micro-ORM acts as a simple interface between the IDE and the database, but lacks a lot of the more powerful and bulkier features that come with typical ORMs, such as automatic object mapping. Micro-ORMs provide the barebones needed to perform its function, and usually involves much more actual writing of SQL code than regular ORMs.[1]

Entity Framework Core

Entity Framework Core is a Microsoft ORM that is built to be used with .NET Core. Entity Framework Core mainly focuses on a code-first approach, whereby C# classes are written and then migrated to a database in an automated fashion.[2]

[1] https://dapper-tutorial.net/dapper
[2] https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview

License

Dapper

Dapper uses the Apache 2.0 license which among other things, gives permission for Dapper to be used in commercial products, free of charge.[3]

Entity Framework Core

Entity Framework Core uses the Apache 2.0 license which among other things, gives permission for Entity Framework Core to be used in commercial products, free of charge.[4]

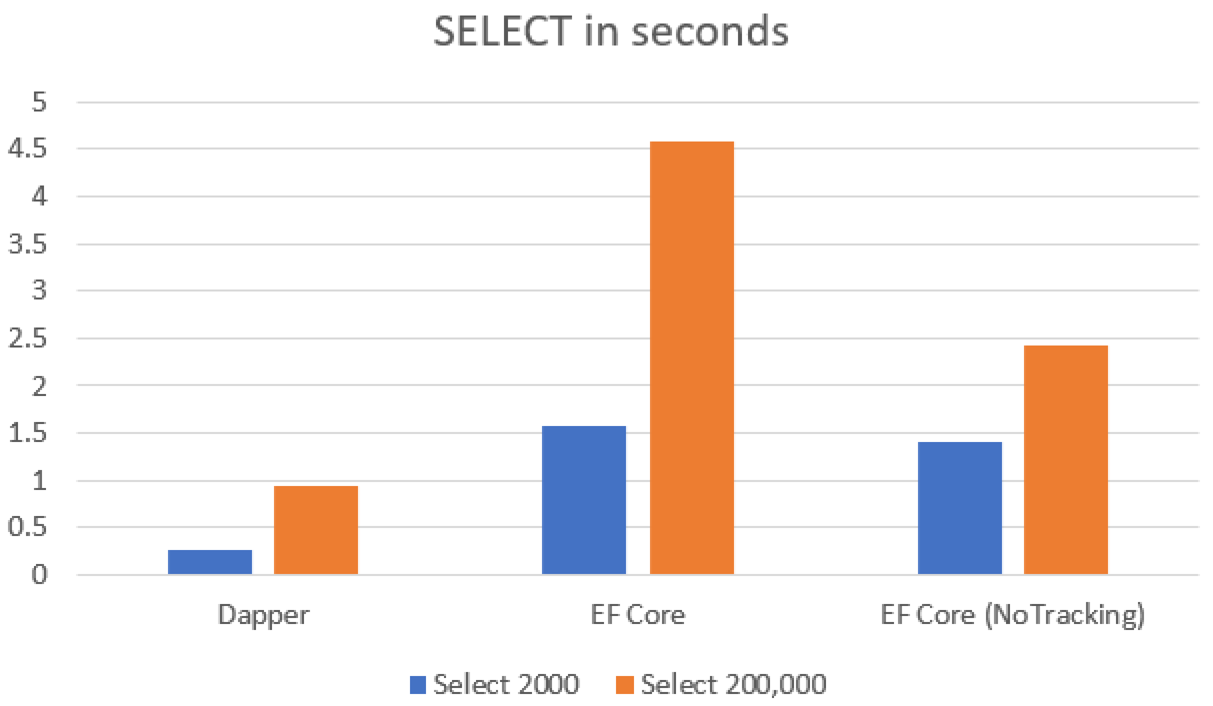
[3] https://github.com/StackExchange/Dapper/blob/main/License.txt

[4] https://github.com/dotnet/efcore/blob/main/LICENSE.txt

Comparisons

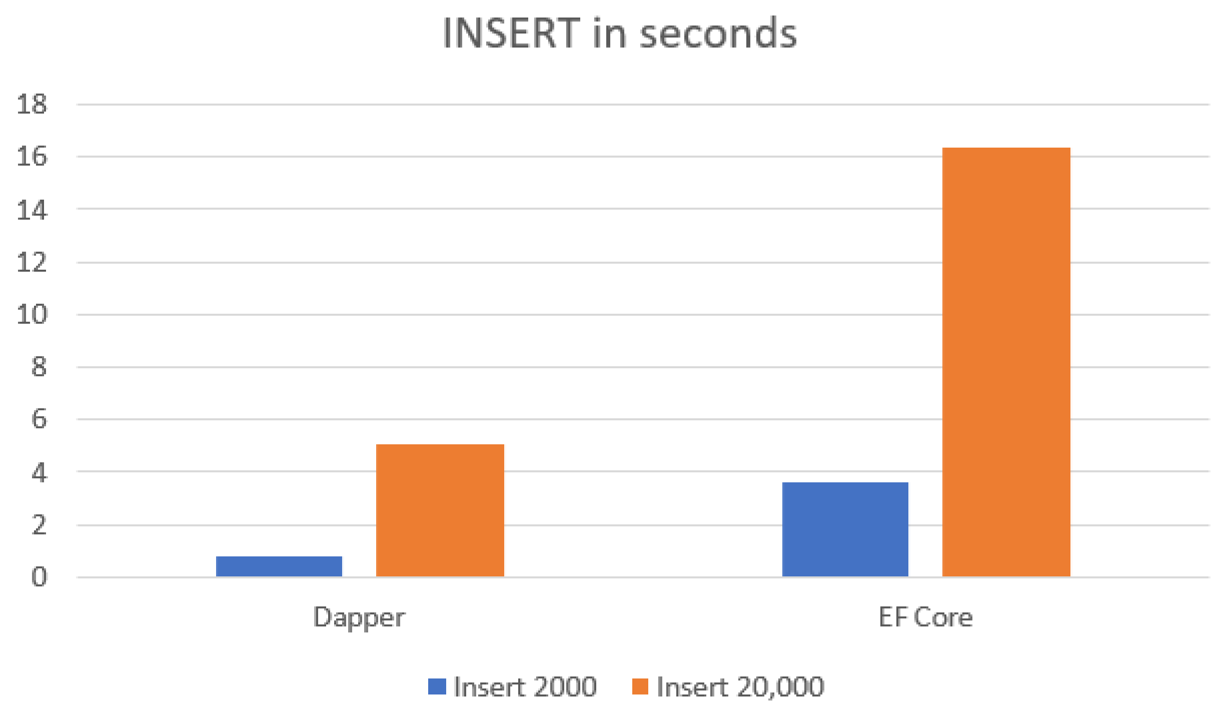
Speed Benchmarks (conducted by Kagawa on Medium)[5]

Select Speed[6]



[5] https://medium.com/@kagacode/speed-benchmark-dapper-vs-ef-core-3-f3777b76dd0
[6] https://medium.com/@kagacode/speed-benchmark-dapper-vs-ef-core-3-f3777b76dd0

Insert Speed[7]



Code-First or Database-First (Which is it more geared toward?)

	Code-First	**Database-First**
Dapper[8]		X
Entity Framework Core[9]	X	

Automatic Capabilities

	Auto-Code Generation	**Automatic Object Creation**
Dapper		
Entity Framework Core[10]	X	X

[7] https://medium.com/@kagacode/speed-benchmark-dapper-vs-ef-core-3-f3777b76dd0

[8] https://visualstudiomagazine.com/articles/2018/03/19/dapper-orm.aspx

[9] https://entityframeworkcore.com/approach-code-first

[10] https://entityframework.net/ef-vs-dapper

Primary Query Method

	Raw SQL	**Linq**
Dapper[11]	X	
Entity Framework Core[12]		X

Bulk Insert Capabilities

	Bulk Insert
Dapper[13]	Requires Dapper Plus
Entity Framework Core[14]	X

Deployment

	Migrations	**Manual**
Dapper		X
Entity Framework Core[15]	X	

[11] https://dapper-tutorial.net/query

[12] https://docs.microsoft.com/en-us/ef/core/querying/

[13] https://dapper-tutorial.net/bulk-insert

[14] https://entityframeworkcore.com/saving-data-bulk-insert

[15] https://abelsquidhead.com/index.php/2017/07/31/deploying-dbs-in-your-cicd-pipeline-with-ef-core-code-first/

Pros and Cons

Dapper

Pros

- Much faster queries than EF Core.
- Requires actual SQL code that makes more complex queries easier and faster.
- Requires actual SQL code that eliminates the problem of accidental data loss.
- Dynamic Parameters that make it easy to prevent SQL injection.[16]
- Database-first approach makes it easy to separate the data access layer from the rest of the application.
- Far less complex than EF Core.

Cons

- Takes longer to create the data access layer due to having to write SQL code.
- Requires knowledge of SQL.
- Does not benefit from code generation and other automatic tools.

Entity Framework Core

Pros

- Faster development time due to auto-code generation, auto-mapping, and auto-object creation.
- Easy queries through Linq.
- Don’t have to know SQL very well due to the abstraction.
- Far superior tooling.
- Easy to make changes through migrations.
- Easy to deploy through migrations.

Cons

- Much slower queries than Dapper.
- Code-first approach makes it harder to separate the data access layer from the front end, reducing modularity and reusability.
- Abstraction leads to not knowing what’s going on under the covers.
- Much more complicated than Dapper and requires a very good understanding of EF Core in order to successfully use.
- Data loss is more of a risk from rollbacks and migrations.[17]

[16] https://www.learndapper.com/parameters

[17] https://livebook.manning.com/book/entity-framework-core-in-action/chapter-11/11

- Some auto-generated code can be problematic if not changed, such as nvarchar types being set to max length.[18]

Discussion

The decision of whether to go with Dapper or Entity Framework Core (EF Core) really boils down to whether we think we would benefit more from a regular ORM or a micro-ORM.

The automatic code-generation capabilities of EF Core would make creating database objects and relationships far faster, however, the slower speed of queries could eventually become a problem down the road if our website gains a substantial amount of users.

EF Core is also very complex and has a substantial learning curve, something that our team does not have as much time for.

Dapper, on the other hand, just requires an understanding of SQL and a small learning curve to learn Dapper library functions. This “closer to the SQL code” approach is much more intuitive to our team and will make it easier for us to create and maintain our databases and data access layer.

While we may not initially see a benefit from the better performance of Dapper, it gives us confidence that our application will be more prepared to handle a large number of users once it goes into production.

Deploying with EF Core migrations does seem like a very convenient process, but the manual process with Dapper leaves much less room for potential bugs.

EF Core’s code-first approach makes it more difficult to have layers of separation between the data access and the front end. EF Core by default requires dependency injection of its services into the frontend, making it harder to keep the data access layer modular and reusable. This is different from Dapper, in which the frontend never requires any knowledge of the ORM being used.

Verdict

Dapper

While EF Core’s impressive features do make life easier, Dapper gives more control in creating and maintaining our data access layer. Dapper is also more beginner friendly, requiring less of a learning curve and is much safer, making dangerous mistakes much less likely to happen. Its faster performance also gives us confidence that our application will be able to handle potential

[18] https://www.sqlservercentral.com/forums/topic/nvarchar4000-and-performance

increases in traffic down the road. Finally, its database-first approach makes it easier to enforce layers of separation, which will help us to better make our code reusable and extensible.

Therefore, our team chooses Dapper as the ORM we would like to request to use.