

FagDag - Mastering EF Core

Learning from common issues when using EF Core

Kenneth Fossen

Let's get going →



Todays topics

- What is EF Core
- EF Core 5 -> 6 -> 7 -> 8
- Pause
- 7 Deadly Sins of EF Core
- Practice / Challenges
- Sources

\$ whoami

```
1 $ whoami
2 {
3     name: Kenneth Fossen,
4     dep: Dragefjellet@Bouvet,
5     email: kenneth.fossen@bouvet.no,
6     edu: [
7         Bachelor i Datatryggleik
8         Master i Programvare Utvikling
9     ],
10    work: [
11        Helse Vest IKT Drift & Sikkerhet
12    ]
13    current: [
14        project: {
15            name: CommonLibrary@Spine
16        },
17        Software Developer,
18        Security Champion,
19        #rustaceans
20    ],
21 }
```



Overview

1. FagDag - Mastering EF Core
2. Todays topics
3. \$ whoami
4. Overview
5. What is EF Core
6. EF Core Changes
7. PAUSE
8. 7 Deadly Sins
9. TIPS
10. Practice / Challenges
11. Source
12. Videos

What is EF Core

EF Core *can* serve as a object-relational-mapper (O/RM)

- Enabling .NET developers to work with a database using .NET Objects
- Eliminates the need for most of the data-access code that typically need to be written.
- It gives you migrations that allow evolving the database as the model changes.

C# Code

```
1  public class BloggingContext : DbContext
2  {
3      public DbSet<Blog> Blogs { get; set; }
4      protected override void OnConfigure(DbContextOptionsBuilder o) {
5          o.UseSqlServer("connectionString");
6      }
7  }
8  public class Blog
9  {
10     public int BlogId { get; set; }
11     public string Url { get; set; }
12     public int Rating { get; set; }
13 }
```



Database table

BlogId	Url	Rating
1	http://blog.kefo.no	5

LINQ to SQL

```
1 // C# Code
2 using var db = new BloggingContext();
3
4 var blogs = db
5     .Blogs
6     .Where(b => b.Rating > 3)
7     .OrderBy(b => b.Url)
8     .ToList(); // Query and fetch data into memory
9
10 // SQL OUTPUT
11 SELECT a."BlogId", a."Url", a."Rating"
12 FROM "Blogs" as a
13 WHERE a."Rating" > 3
14 ORDER BY a."Url"
```

EF

- EF6 is a ORM for .NET Framework
- Supports also .NET Core
- No longer developed

EF Core

- EF Core is modern ORM for .NET
- Support for LINQ
- Change Tracking
- Updates
- Schema Migrations

Dapper

- A simple object mapper for .NET
- Known to be the fastest
- Low memory footprint (compared to EF Core)
- Complete control over your SQL Queries
- Go-to solution when performance is critical

Dapper - a simple object mapper for .NET

Dapper - a simple object mapper for .Net

[View on GitHub](#)

Dapper - a simple object mapper for .NET

Overview

A brief guide is available [on github](#)

Questions on Stack Overflow should be tagged `dapper`

Project: Dapper vs EF Core

```
1  var sql = @"
2      SELECT l.[Id]
3          ,l.[Name]
4          ,l.[Description]
5          ,l.[IsCaseSensitive]
6          ,l.[IsForeignObject]
7          ,l.[IsGlobal]
8          ,scopeType.[Name]  as ScopeType
9          ,case when l.[NamesCasing] = {=UpperCase} then 1 else 0 end as AreNamesUpperCase
10         ,l.[Alias]
11         ,l.[AttachmentKey]
12     FROM [dbo].[Library] l
13     LEFT JOIN [dbo].[Library] scopeType ON scopeType.Id = l.ScopeTypeId
14     WHERE l.IsValid = 1
15     ORDER BY l.[Name]
16     SELECT DISTINCT lt.LibraryId, t.Name
17         FROM LibraryTag lt
18         JOIN Tag t ON t.Id = lt.TagId
19         JOIN Library l ON l.Id = lt.LibraryId
20     WHERE l.IsValid = 1
21     SELECT DISTINCT lg.LibraryId, g.Name
22         FROM LibraryAccessGroup lg
```

```
1 var libraryListItems = _context.  
2     Library  
3     .TagWith("LibraryList LINQ")  
4     .AsNoTracking()  
5     .Where(library => library.IsValid)  
6     .OrderBy(library => library.Name)  
7     .Select(library => new LibraryListItem  
8 {  
9     Id = library.Id,  
10    Name = library.Name,  
11    Desc = library.Description,  
12    Alias = library.Alias,  
13    IsGlobal = library.IsGlobal,  
14    ScopeType = library.ScopeType.Name,  
15    IsForeignObject = library.IsForeignObject,  
16    AreNamesUpperCase = library.IsCaseSensitive,  
17    AttachmentKey = library.AttachmentKey,  
18    Tags = library  
19        .LibraryTags  
20        .Select(tag => tag.Tag.Name)  
21        .ToList(),  
22    AccessGroups = library  
23        .LibraryAccessGroups  
24        .Select(group => group.AccessGroup.Name)  
25        .ToList()  
26});  
27  
28 return Ok(libraryListItems);
```

EF Core Changes

5 -> 6 -> 7 -> 8



EF Core 5

⚠️ Query Strategy Changed ⚠️

- Performance
- Complex Queries that was fast **may** be slow

Lets study this code

```
1  var artists = context  
2      .Artists  
3      .Include(e => e.Albums)  
4      .ToList();
```

Split Query

```
1 SELECT a."Id", a."Name"  
2 FROM "Artists" AS a  
3 ORDER BY a."Id"  
4  
5 SELECT a0."Id", a0."ArtistId", a0."Title", a."Id"  
6 FROM "Artists" AS a  
7 INNER JOIN "Album" AS a0 ON a."Id" = a0."ArtistId"  
8 ORDER BY a."
```

Combined Query

```
1 SELECT a."Id", a."Name",  
2     a0."Id", a0."ArtistId", a0."Title"  
3 FROM "Artists" AS a  
4 LEFT JOIN "Album" AS a0 ON a."Id" = a0."ArtistId"  
5 ORDER BY a."Id", a0."Id"
```

3 Options

```
1 var blogs = context
2   .Blogs
3   .Include(blog => blog.Posts)
4   .AsSplitQuery()
5   .ToList();
```

```
1 var blogs = context
2   .Blogs
3   .Include(blog => blog.Posts)
4   .AsSingleQuery()
5   .ToList();
```

```
1 protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
2 {
3     optionsBuilder.UseSqlServer(
4         @"Server=(LocalDB)\mssqllocaldb;Database=EFQuerying;Trusted_Connection=True",
5         o => o.UseQuerySplittingBehavior(QuerySplittingBehavior.SplitQuery)
6     );
7 }
```

From my project: EF optimization to handle timeout

Old strategy fetched CableSpecWire for facility ROM in 11 minutes, using 90% of available compute power.

```
1  if (specification is CodeSetWithReferencedCodesSpecification)
2  {
3      query = query.AsSplitQuery(); // Optimization for large sets
4 }
```

With this small change the same query takes 6 seconds.

EF Core 6

- SQL Server temporal tables
- Migration Bundles
- Pre-convention model configuration
- Compiled models
- Performance
 - 70% better query performance than EF Core 5.0
 - 31% faster executing untracked queries
 - Heap allocations have been reduced by 43%

EF Core 7

- JSON Columns
- ExecuteUpdate and ExecuteDelete (BulkUpdates)
 - Improved commands are sent to the database, can only act on a single table
 - ⚠ May cause out of sync with the Entity Tracker ⚠
- Faster `SaveChanges`
 - 4x times faster than EF Core 6
 - Has fewer roundtrips to the database when saving
- Stored Procedure Mapping
 - You can now map in the `modelBuilder` of the `DbContext` for an Entity what stored procedures should be used, and what parameters to submit to them.

EF Core 7 Continue

- Query Enhancement

`.GroupBy()` can now be the final operator in a query.

`.GroupBy(s => s.Author)` group on a entity type

`.GroupJoin()` can be used as the final operator

`Contains()` now accepts `IReadOnlyCollections`

- Model Building Enhancement

Indices can be ascending or descending `.IsDescending()`

```
modelBuilder
    .Entity<Post>()
    .HasIndex(post => post.Title)
    .IsDescending();
```

EF Core 7: Breaking change

Database Connection Strings

`Encrypt` defaults to `true` for SQL Server connections

- Consequences, you need a **valid** certificate on the SQL Server.
- The client must trust this certificate
- 2 ways to mitigate this:
 - `TrustServerCertificate=True`
 - `Encrypt=False`

EF Core 8

It will be released this coming November together with .NET8.

- Raw SQL queries for unmapped types

The Query `.SqlQuery<T>("SELECT * FROM T")` is parameterized, SQLi safe.

- DateOnly/TimeOnly support for SQL Server

You can use DateOnly / TimeOnly types from .NET6. They are now introduced in EF Core 8.

💡 There is also a package to use DateOnly / TimeOnly if you need it EF Core 6 / EF Core 7.

NuGet `ErikEJ.EntityFrameworkCore.SqlServer.DateOnlyTimeOnly`

- Close to Dapper Performance and Memory Consumption

EF Core 8: Breaking changes

- The scaffolding of a database now translates to DateOnly and TimeOnly for `date` and `time`.
- `Contains` in LINQ queries may stop working on older SQL Server versions.
- Only supports SQL Server 2014 above

PAUSE

After pause

7 Deadly Sins

Practice

Challenges



7 Deadly Sins

Deadly Sin: 1

Casting `IQueryable` to `Enumerable`

Examples

- `.ToList()`
- `.AsEnumerable()`

⚠ Will fetch all the data! ⚠

Code Example

```
1  public IEnumerable<Sales> GetSales() {
2      return SalesDbContext.Sales;
3  }
4
5  public int CountSalesInDb() {
6      return GetSales().Count();
7      // SELECT * FROM Sales
8      // in memory count the collection
9  }
10 public int CountSalesInDb2() {
11     return GetSales2().Count();
12     // SELECT COUNT(*) FROM Sales;
13 }
14
15 private IQueryable<Sale> GetSales2() {
16     return SalesDbContext.Sales;
17 }
```

Deadly Sin: 2

Not using `.AsNoTracking()`

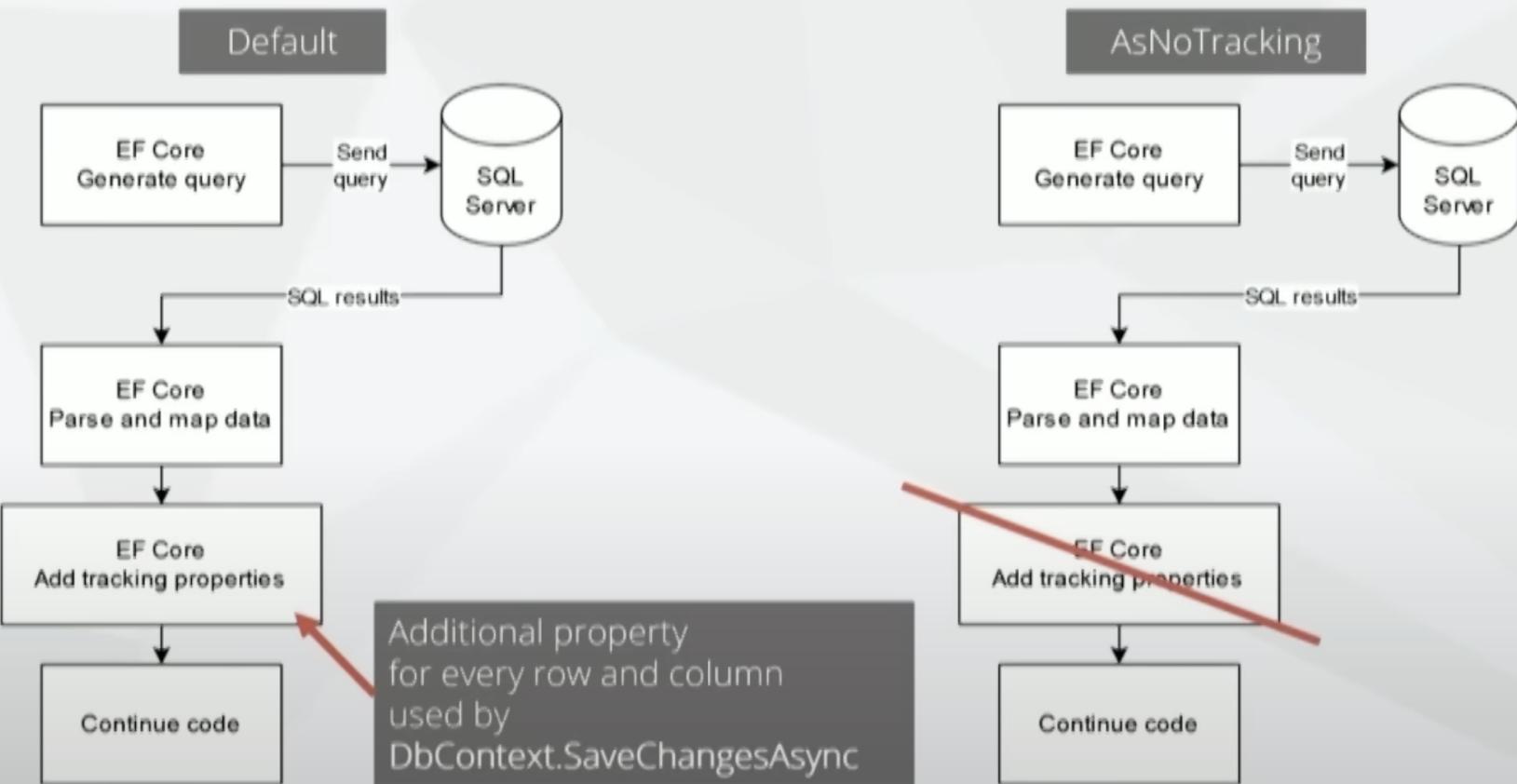
If you are only reading from the database, there there is no need to track the entity.

Perfomance 

- Can use ~4x less memory
- Can be ~5x faster

 Not for SQLite DB Provider 

#2 What is happening?



Deadly Sin: 3

Explicit Joins

```
.Include(x => x.Customers)
```

1. They are always included
2. We get all their columns
3. Forget to remove them

Deadly Sin: 4

Getting all the columns

~2.5 faster with `Select()` and implicit joins.

Example

```
1  var explicitQuery = _dbContext.Sales
2      .AsNoTracking()
3      .TagWithContext()
4      .Include(x => x.SalesPerson)
5      .Where(x => x.SalesPersonId == 1);
6
7  var implicitQuery = _dbContext.Sales
8      .AsNoTracking()
9      .Where(x => x.SalesPersonId == 1)
10     .Select(x => new SalesWithSalesPerson {
11         CustomerId = x.CustomerId,
12         SalesId = x.SalesPersonId,
13         ProductId = x.ProductId,
14         Quantity = x.Quantity
15         SalesPersonId = x.SalesPersonId,
16         SalesPersonFirstName = x.SalesPerson.FirstName,
17         SalesPersonLastName = x.SalesPersons.LastName
18     });

```

Deadly Sin: 5

No pagination

- Fetch all data
- Process in memory

```
1  var query = context
2    .Sales
3    .AsNotTracking()
4    .Where(x => x.SalesPersonId == salesPersonId);
5
6  var result = await query.ToListAsync(ct);
7  int count = result.Count;
8
9  result = result
10   .Skip(page * pageSize)
11   .Take(pageSize);
12
13 return (count, result);
```

```
1  var query = context
2    .Sales
3    .AsNotTracking()
4    .Where(x => x.SalesPersonId == salesPersonId);
5
6  int count = await query.CountAsync(ct);
7
8  query = query
9    .Skip(page * pageSize)
10   .Take(pageSize);
11
12 var result = await query.ToListAsync(ct);
13 return (count, result);
```

Deadly Sin: 6

Not using Cancellation Tokens

- Will stop the Query on the SQL Server also

```
1  var query = context
2      .Sales
3      .AsNotTracking()
4      .Where(x => x.SalesPersonId == salesPersonId);
5
6  int count = await query.CountAsync(ct);
7
8  query = query
9      .Skip(page * pageSize)
10     .Take(pageSize);
11
12 var result = await query.ToListAsync(ct);
13 return (count, result);
```

Deadly Sin: 7

Inefficient Updates and Deletes

`foreach` will generate n SQL statements

`foreach`

```
1 foreach (var blog in context.Blogs.Where(b => b.Rating < 3))
2 {
3     context.Blogs.Remove(blog);
4 }
5
6 context.SaveChanges();
```

`.ExecuteDelete()`

```
1 context.Blogs
2     .Where(b => b.Rating < 3)
3     .ExecuteDelete();
4
5 // SQL
6 DELETE FROM [b]
7 FROM [Blogs] AS [b]
8 WHERE [b].[Rating] < 3
```

TIPS

- `.TagWith("Fetching all users")`
- `.Select()` does not need `.AsNoTracking()`
- Use `-Async` version of the operator e.g (`ToListAsync()`)
- `Select` to only fetch the right columns
- An Index will not be used if there are operations on a Column

```
1 options.EnableDetailedErrors()
2 options.EnableSensitiveDataLogging()
3 options.ConfigureWarnings(wa =>
4     was.Log( new EventId[] {
5         CoreEventId.FirstWithoutOrderByAndFilterWarning,
6         CoreEventId.RowLimitingOperationsWithoutOrderByWarning
7     })
8 )
```

TIPS Cont.

If you use a lot of Include or Join, a view might be a good idea.

```
1 modelBuilder.Entity<PostWithPostType>(e => entity  
2     .ToView("PostWithType"));
```

Use Data Annotation to limit e.g. size of your types e.g. `string`

```
1  
2     public string Firstname { get; set; }  
3  
4     // SQL produced  
5     Firstname nvarchar(max) -- (2GB)  
6  
7     [StringLength(64)] // nvarchar(64)  
8     public string Name { get; set; }  
9  
10    [MinLength(5), MaxLength(12)]  
11    public string Password { get; set; }
```

Practice / Challenges

<https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>

<https://learn.microsoft.com/en-us/training/modules/persist-data-ef-core/>

<https://learn.microsoft.com/en-us/training/modules/build-web-api-minimal-database/>

<https://github.com/dotnet/EntityFramework.Docs/tree/main/samples/core/Querying/RelatedData>

Expert Challenge

<https://github.com/StefanTheCode/OptimizeMePlease>

Source

EF Core Doc: <https://learn.microsoft.com/en-us/ef/core/>

Book: <https://www.manning.com/books/entity-framework-core-in-action-second-edition>

SSW EF Core: <https://ssw.com.au/rules/rules-to-better-entity-framework/>

SSW Linq: <https://ssw.com.au/rules/rules-to-better-linq/>

Single vs Split Queries: <https://learn.microsoft.com/en-us/ef/core/querying/single-split-queries>

Sealed Classes: <https://code-maze.com/improve-performance-sealed-classes-dotnet/>

N+1 Issue : <https://levelup.gitconnected.com/the-hidden-performance-killer-in-ef-core-understanding-and-avoiding-the-n-1-query-issue-ce105c6a14e9>

Videos

