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

```
$ whoami
{
     name: Kenneth Fossen,
     dep: Dragefjellet@Bouvet,
     email: kenneth.fossen@bouvet.no,
     edu: [
         Bachelor i Datatryggleik
         Master i Programvare Utvikling
     ],
     work: [
         Helse Vest IKT Drift & Sikkerhet
     ]
     current: [
          project: {
              name: CommonLibrary@Spine
          },
          Software Developer,
          Security Champion,
          #rustaceans
      ],
 }
```

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

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    protected override void OnConfigure(DbContextOptionsBuilder o) {
      o.UseSqlServer("connectionString");
    }
}
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
}
```

Database table

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

LINQ to SQL

```
// C# Code
using var db = new BloggingContext();

var blogs = db
    .Blogs
    .Where(b => b.Rating > 3)
    .OrderBy(b => b.Url)
    .ToList(); // Query and fetch data into memory

// SQL OUTPUT
SELECT a."BlogId", a."Url", a."Rating"
FROM "Blogs" as a
WHERE a."Rating" > 3
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
- Know to be the fastest
- Low memory footprint (compared to EF Core)
- Complete control over your SQL Queries
- Goto solution when performance is critical

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

```
var sql = @"
    SELECT l.[Id]
          ,l.[Name]
          ,l.[Description]
          ,l.[IsCaseSensitive]
          ,l.[IsForeignObject]
          ,l.[IsGlobal]
          ,scopeType.[Name] as ScopeType
          ,case when l.[NamesCasing] = {=UpperCase} then 1 else 0 end as AreNamesUpperCase
          ,l.[Alias]
          ,l.[AttachmentKey]
      FROM [dbo].[Library] l
      LEFT JOIN [dbo].[Library] scopeType ON scopeType.Id = l.ScopeTypeId
    WHERE l.IsValid = 1
    ORDER BY l.[Name]
    SELECT DISTINCT lt.LibraryId, t.Name
      FROM LibraryTag lt
      JOIN Tag t ON t.Id = lt.TagId
      JOIN Library l ON l.Id = lt.LibraryId
    WHERE l.IsValid = 1
    SELECT DISTINCT lg.LibraryId, g.Name
      FROM LibraryAccessGroup lg
```

```
var libraryListItems = _context.
  Library
  .TagWith("LibraryList LINQ")
  .AsNoTracking()
  .Where(library => library.IsValid)
  .OrderBy(library => library.Name)
  .Select(library => new LibraryListItem
  {
      Id = library.Id,
      Name = library.Name,
      Desc = library.Description,
      Alias = library.Alias,
      IsGlobal = library.IsGlobal,
      ScopeType = library.ScopeType.Name,
      IsForeignObject = library.IsForeignObject,
      AreNamesUpperCase = library.IsCaseSensitive,
      AttachmentKey = library.AttachmentKey,
      Tags = library
          .LibraryTags
          .Select(tag => tag.Tag.Name)
          .ToList(),
      AccessGroups = library
          .LibraryAccessGroups
          .Select(group => group.AccessGroup.Name)
          .ToList()
  });

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

```
var artists = context
    .Artists
    .Include(e => e.Albums)
    .ToList();
```

Split Query

```
SELECT a."Id", a."Name"
FROM "Artists" AS a
ORDER BY a."Id"

SELECT a0."Id", a0."ArtistId", a0."Title", a."Id"
FROM "Artists" AS a
INNER JOIN "Album" AS a0 ON a."Id" = a0."ArtistId"
ORDER BY a."
```

Combined Query

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

3 Options

```
var blogs = context
    .Blogs
    .Include(blog ⇒ blog.Posts)
    .AsSplitQuery()
    .ToList();
```

```
var blogs = context
    .Blogs
    .Include(blog ⇒ blog.Posts)
    .AsSingleQuery()
    .ToList();
```

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer(
      @"Server=(localdb)\mssqllocaldb;Database=EFQuerying;Trusted_Connection=True",
      o ⇒ o.UseQuerySplittingBehavior(QuerySplittingBehavior.SplitQuery)
    );
}
```

From my project: EF optimization to handle timeout

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

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

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% afaster 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 parameteriezed, 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 `IEnumerable`

Examples

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

⚠ Will fetch all the data! ⚠

Code Example

```
1   public IEnumerable<Sales> GetSales() {
2     return SalesDbContex.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()`

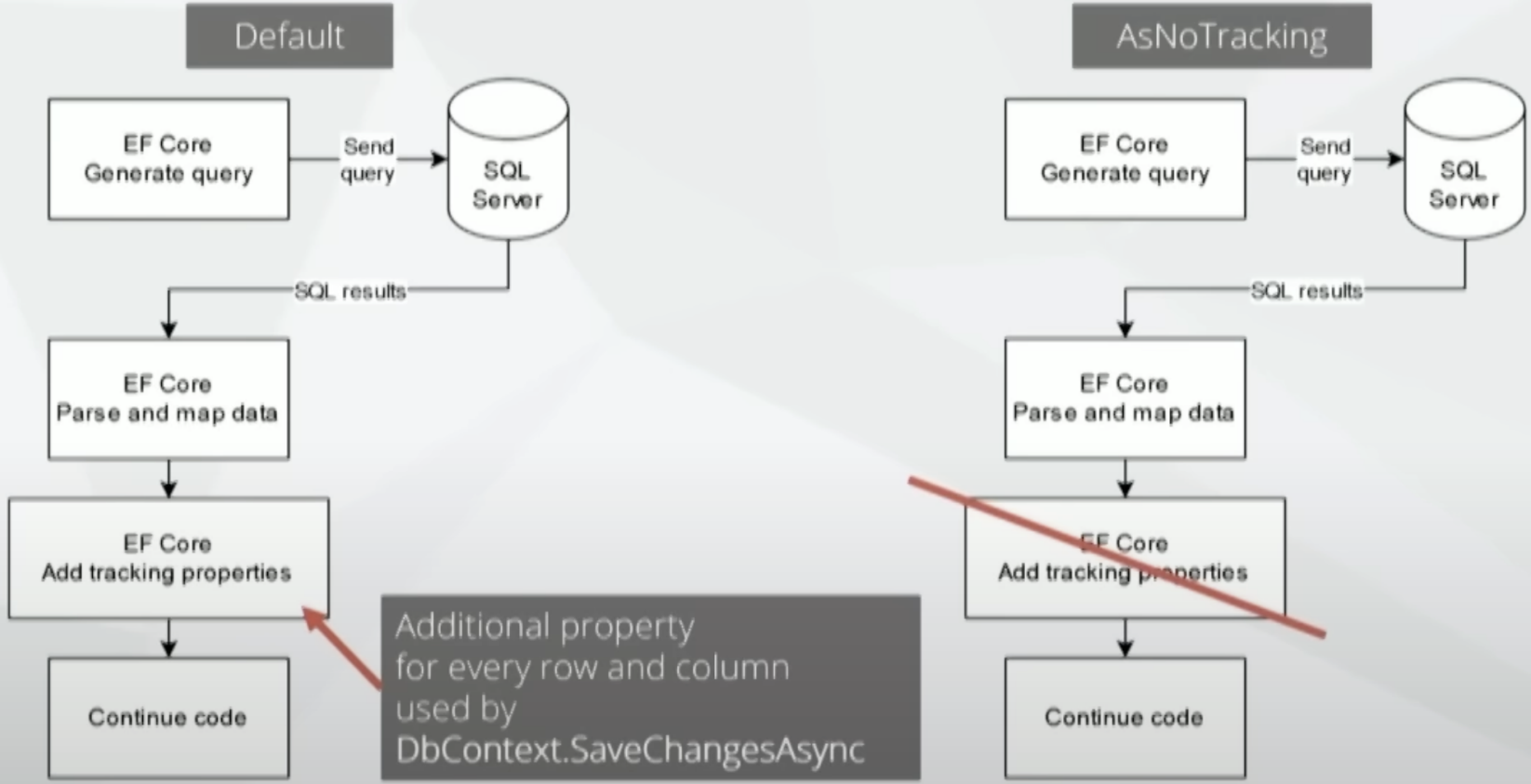
If you are only reading from the database, there there is no need to track the entity.

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

```
var explicitQuery = _dbContext.Sales
    .AsNoTracking()
    .TagWithContext()
    .Include(x => x.SalesPerson)
    .Where(x => x.SalesPersonId == 1);

var implicitQuery = _dbContext.Sales
    .AsNoTracking()
    .Where(x => x.SalesPersonId == 1)
    .Select(x => new SalesWithSalesPerson {
        CustomerId = x.CustomerId,
        SalesId = x.SalesPersonId,
        ProductId = x.ProductId,
        Quantity = x.Quantity
        SalesPersonId = x.SalesPersonId,
        SalesPersonFirstName = x.SalesPerson.FirstName,
        SalesPersonLastName = x.SalesPersons.LastName
    });
```

Deadly Sin: 5

No pagination

- Fetch all data
- Process in memory

```
var query = context
    .Sales
    .AsNotTracking()
    .Where(x => x.SalesPersonId == salesPersonId);

var result = await query.ToListAsync(ct);
int count = result.Count;

result = result
    .Skip(page * pageSize)
    .Take(pageSize);

return (count, result);
```

```
var query = context
    .Sales
    .AsNotTracking()
    .Where(x => x.SalesPersonId == salesPersonId);

int count = await query.CountAsync(ct);

query = query
    .Skip(page * pageSize)
    .Take(pageSize);

var result = await query.ToListAsync(ct);
return (count, result);
```

Deadly Sin: 6

Not using Cancellation Tokens

- Will stop the Query on the SQL Server also

```
var query = context
    .Sales
    .AsNotTracking()
    .Where(x => x.SalesPersonId == salesPersonId);

int count = await query.CountAsync(ct);

query = query
    .Skip(page * pageSize)
    .Take(pageSize);

var result = await query.ToListAsync(ct);
return (count, result);
```

Deadly Sin: 7

Inefficient Updates and Deletes

`foreach` will generate `n` SQL statements

`foreach`

```
foreach (var blog in context.Blogs.Where(b ⇒ b.Rating < 3))
{
    context.Blogs.Remove(blog);
}

context.SaveChanges();
```

`.ExecuteDelete()`

```
context.Blogs
  .Where(b ⇒ b.Rating < 3)
  .ExecuteDelete();

// SQL
DELETE FROM [b]
FROM [Blogs] AS [b]
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

```
options.EnableDetailedErrors()
options.EnableSensetiveDataLoggging()
options.ConfigureWarnings(wa ⇒
  was.Log( new EventId[] {
    CoreEventId.FirstWithOutOrderByAndFilterWarning
    CoreEventId.RowLimitingOperationsWithoutOrdferByWarning
  })
)
```

TIPS Cont.

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

```
modelBuilder.Entity<PostWithPostType>(e ⇒ entity
      .ToView("PostWithType"));
```

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

```

public string Firstname { get; set; }

// SQL produced
Firstname nvarchar(max) -- (2GB)

[StringLength(64)] // nvarchar(64)
public string Name { get; set; }

[MinLength(5), MaxLength(12)]
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


{ } NDC Conferences
Common mistakes i...


{ } NDC Conferences
Entity Framework (C...
Dan Malon


EF Core Performanc...
233x
FASTER