

# Common mistakes in



# EF Core

NDC London 2024

Join the Conversation @SSW\_TV @jernej\_kavka #EFCore



# Jernej Kavka (JK)

SSW Solution Architect



[@Jernej\\_kavka](https://twitter.com/Jernej_kavka)



[github.com/jernejk](https://github.com/jernejk)



[jkdev.me](https://jkdev.me)



[linkedin.com/in/jernejkavka/](https://linkedin.com/in/jernejkavka/)



Brisbane Full Stack User Group



Host @ Global AI The Podcast

.NET and EF Core dev

Microsoft AI MVP and AI generalist

Join the Conversation @SSW\_TV @jernej\_kavka #EFCore





# Agenda



JAN

FEB

MAR

APR

MAY

MONTHLY TIP

SUNDAY

MONDAY

TUESDAY

PLAN SMART.

FILL OUT YOUR  
WEEKLY AGENDA  
NOTING WHEN YOU  
WILL DO WHAT AND  
HOW. THIS WILL  
KEEP YOU ON TRACK  
TO ATTAINING YOUR  
GOAL.

Intro

NOTES

The 7 Deadly Sins

Mini Best Practices

# We have lots of DB engines data access landscape



SQL Server



Cosmos DB



PostgreSQL



MySQL

# Various ORM



EF Core (multiple DBs)



Dapper (multiple DBs)



Marten (PostgreSQL)



Cosmos DB SDK

# Why I ❤️ EF Core

- ✓ Great balance

  - ✨ Features


  - 🤸 Flexibility

  - ⚡ Performance

- ✓ Awesome for relational DBs

- ✓ Built-in migrations, DB scaffolding and other tooling

# Avoid the pain

- The EF Core issues that are solved
- Simplifying your code
- Benchmark your code with tests
- The horror I have seen... 

# Benchmark tools

- BenchmarkDotNet 

Memory consumption and time difference

- Bombardier 

Simple CLI load testing tool



# My testing environment

- .NET 7 with EF Core 7
- My PC
  - CPU: Ryzen 9 5900X
    - Cores: 12
    - Threads: 24
  - Ram: 32GB

The screenshot shows the CPU-Z application window. The 'CPU' tab is selected, displaying the following information:

- Processor:**
  - Name: AMD Ryzen 9 5900X
  - Code Name: Vermeer
  - Package: Socket AM4 (1331)
  - Technology: 7 nm
  - Max TDP: 105.0 W
  - Core Voltage: 1.224 V
  - Specification: AMD Ryzen 9 5900X 12-Core Processor
  - Family: F
  - Model: 1
  - Stepping: 0
  - Ext. Family: 19
  - Ext. Model: 21
  - Revision: B0
  - Instructions: MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A, x86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA
- Clocks (Core #0):**
  - Core Speed: 3592.72 MHz
  - Multiplier: x 36.0
  - Bus Speed: 99.80 MHz
  - Rated FSB: (empty)
- Cache:**
  - L1 Data: 12 x 32 KBytes, 8-way
  - L1 Inst.: 12 x 32 KBytes, 8-way
  - Level 2: 12 x 512 KBytes, 8-way
  - Level 3: 2 x 32 MBytes, 16-way

At the bottom, the 'Selection' dropdown is set to 'Socket #1'. A red box highlights the 'Cores' field (12) and the 'Threads' field (24). The CPU-Z logo and version 'Ver. 1.97.0.x64' are visible at the bottom left, and 'Tools', 'Validate', and 'Close' buttons are at the bottom right.

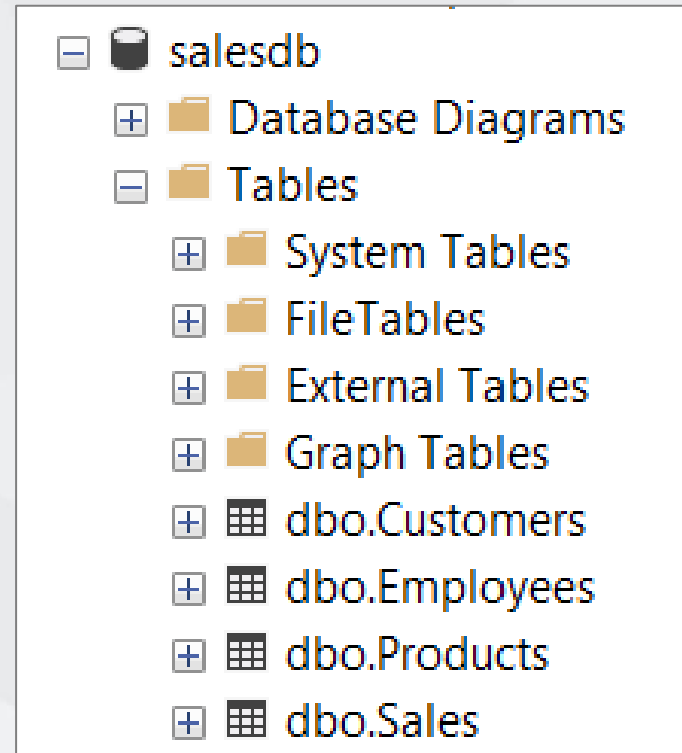
# Test data



- 7 million sales
- 504 products
- 23 employees
- 20,000 customers



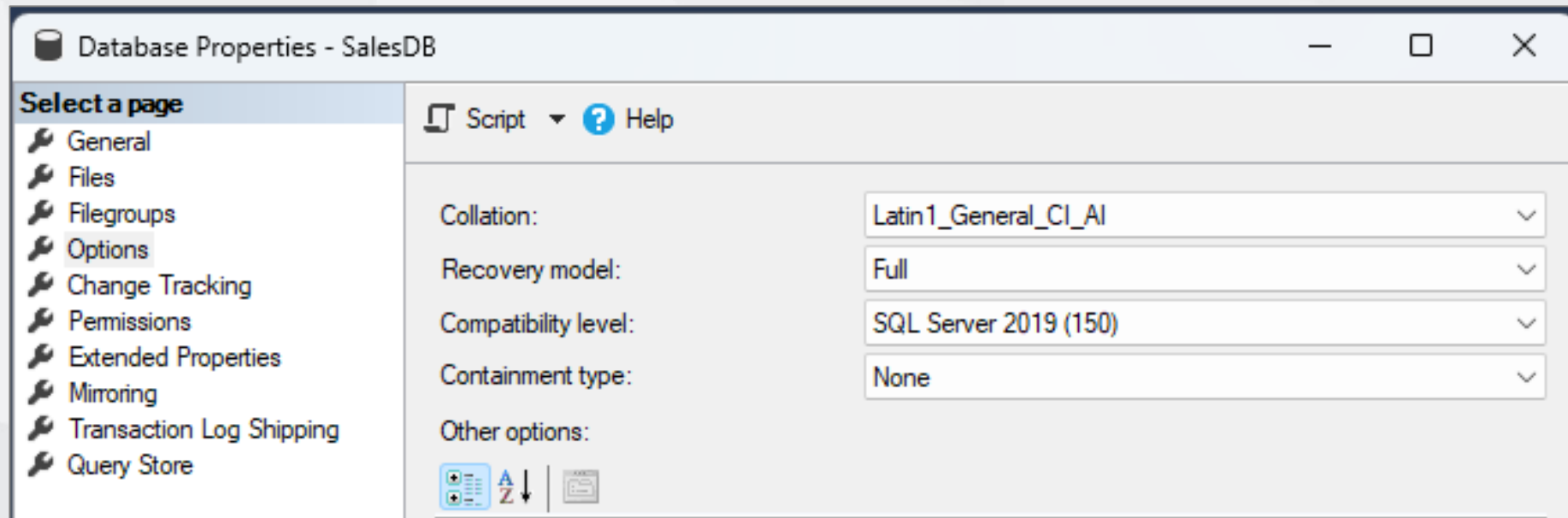
[sqlskills.com/sql-server-resources/sql-server-demos/](https://sqlskills.com/sql-server-resources/sql-server-demos/)



# ⚠ Update to SQL Server 2016+

Compatibility level: 130+ (SQL Server 2016+)

```
ALTER DATABASE SalesDB SET COMPATIBILITY_LEVEL = 150;
```





The 7 Deadly Sins 🍏 🐍



# The 7 Deadly EF Core Sins (easy to hard)

1. Casting **IQueryable** -> **IEnumerable**
2. Not using **AsNoTracking** (when appropriate)
3. Explicit joins
4. Getting all the columns
5. No pagination
6. Non-cancellable queries
7. Inefficient updates/deletes





# "How bad can it be?"

(you never want the answer to this 🤖)

- Literally everyone before discovering the horrifying truth

Join the Conversation @SSW\_TV @jernej\_kavka #EFCore

# #1 IQueryable -> IEnumerable

[Benchmark]

0 references | 0 changes | 0 authors, 0 changes

```
public int NaiveCount()
{
    return SalesDbContext.Sales
        .ToList()
        .Count;
}
```

[Benchmark]

0 references | 0 changes | 0 authors, 0 changes

```
public int QuerableToEnumerableCount()
{
    IEnumerable<Sale> sales = SalesDbContext.Sales;
    return sales.Count();
}
```

[Benchmark]

0 references | 0 changes | 0 authors, 0 changes

```
public int ImplicitEnumerableCount()
{
    return GetSalesEnumerable().Count();
}
```

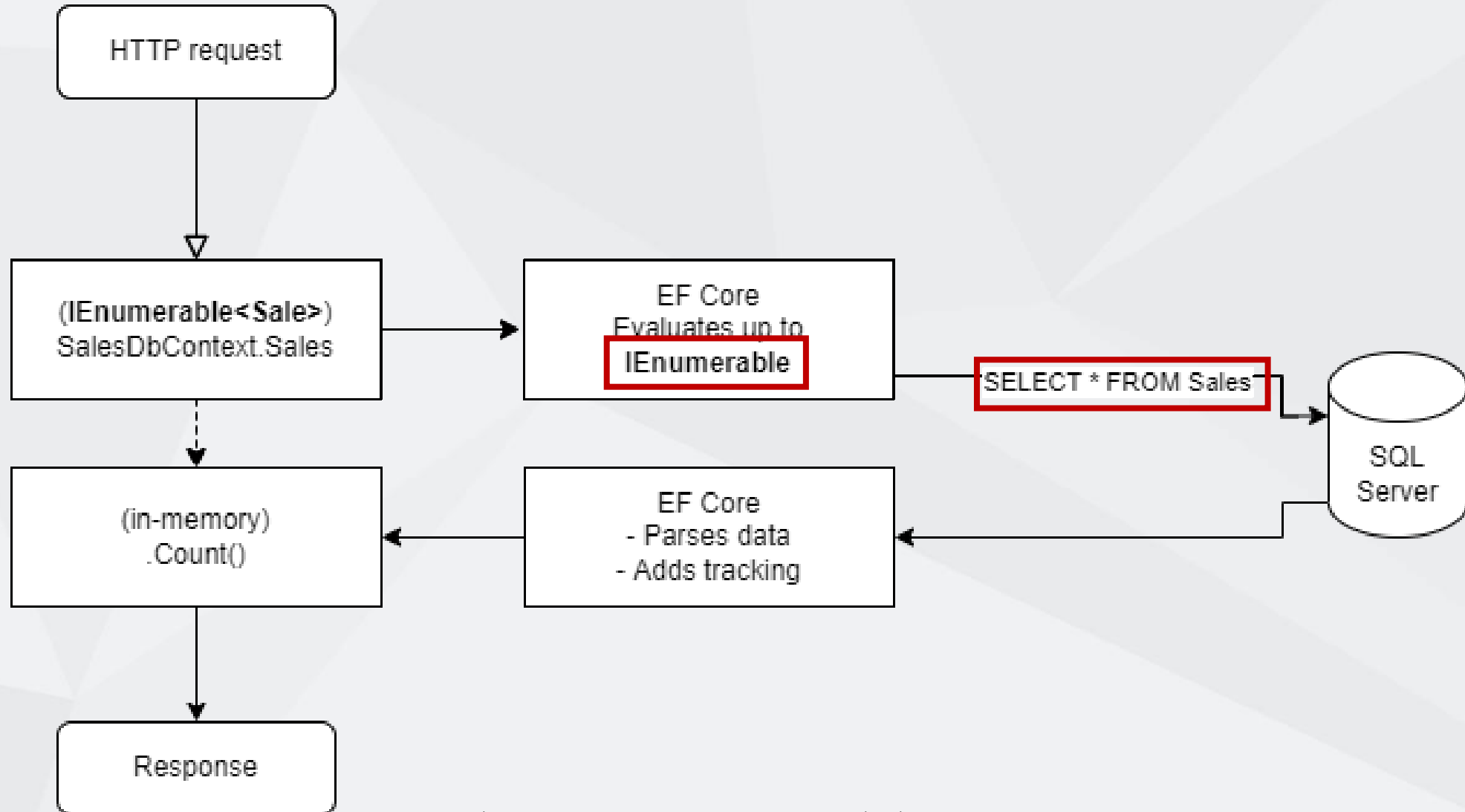
1 reference | 0 changes | 0 authors, 0 changes

```
private IEnumerable<Sale> GetSalesEnumerable()
{
    return SalesDbContext.Sales;
}
```



THEY ARE ALL  
THE SAME

# #1 What is going on?





# #1 IQueryable

```
[Benchmark(Baseline = true)]
```

```
0 references | 0 changes | 0 authors, 0 changes
```

```
public int CountInDb()
```

```
{
```

```
    return GetSales().Count();
```

```
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
```

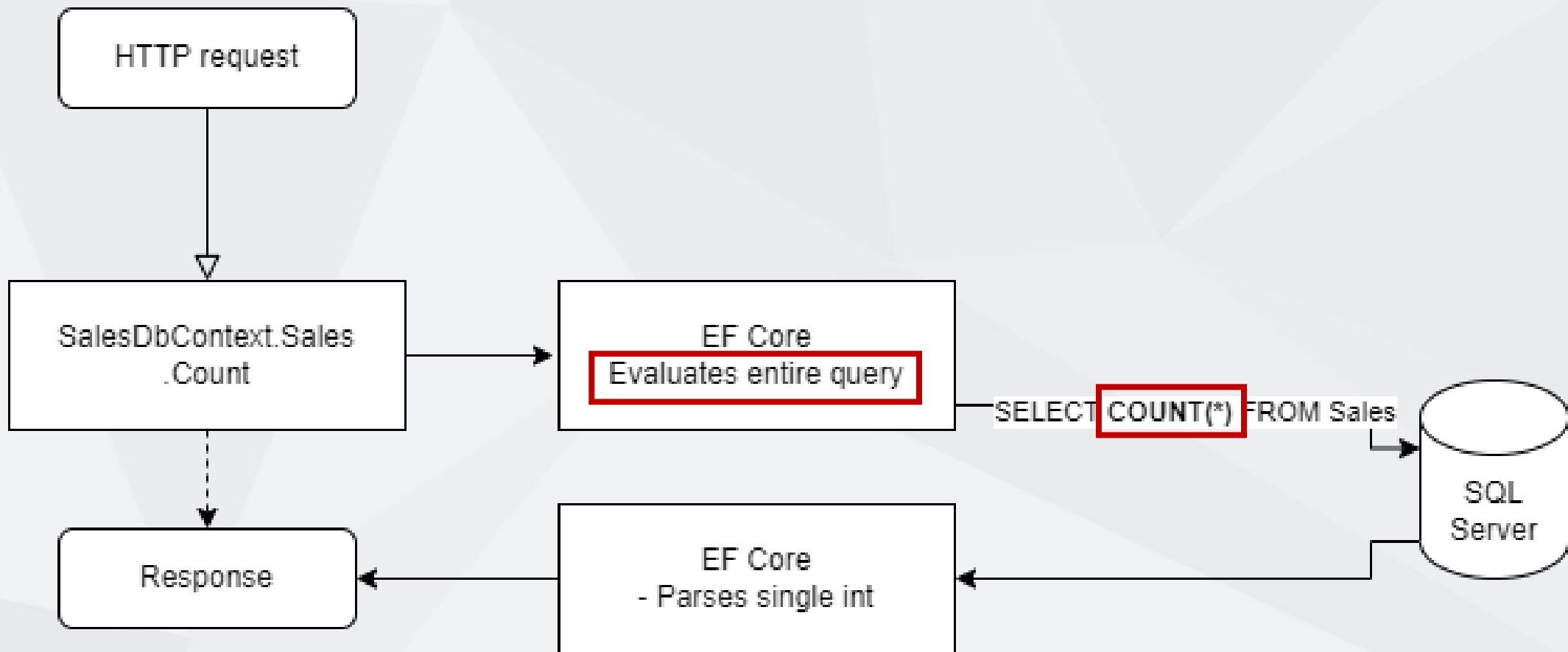
```
private IQueryable<Sale> GetSales()
```

```
{
```

```
    return SalesDbContext.Sales;
```

```
}
```

# #1 How is this better?



IQueryable + EF Core (EF Core LINQ)  
≠  
IEnumerable (Standard LINQ)

# #1 What is the memory impact?

Method	Mean	Error	StdDev	Ratio	RatioSD	Gen 0	Allocated
NaiveCount	3,675.44 ms	69.775 ms	71.654 ms	63.91	1.78	224000.0000	1,967,272 KB
CountInDb	57.52 ms	0.663 ms	0.588 ms	1.00	0.00	-	6 KB

Full 3.5 seconds faster!

64 times faster

~million times  
less memory



# #1 What about load testing?

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesCount/worstCase
Bombarding https://localhost:5001/ExamplesCount/worstCase for 1m0s using 5 connection(s)
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec        0.09       3.10    149.94
Latency      33.36s      7.62s    47.20s
```

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesCount/worstCase
Bombarding https://localhost:5001/ExamplesCount/worstCase for 1m0s using 5 connection(s)
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       38.34      21.26    235.21
Latency     128.69ms     23.25ms    269.06ms
```

~260x faster on average in a load test!

## Memory

Memory usage

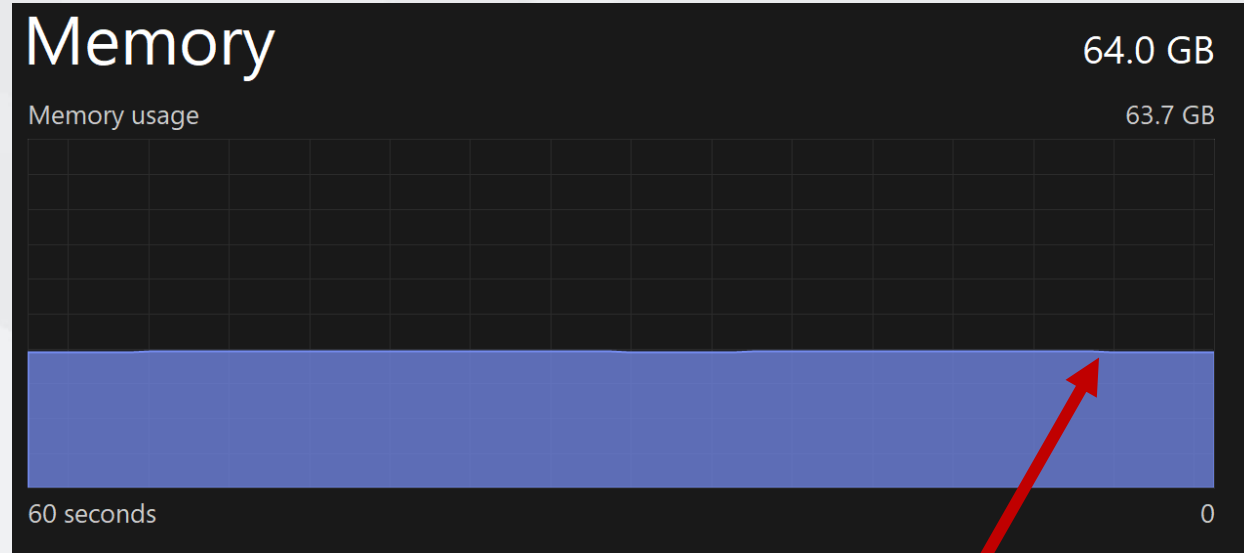
64.0 GB

63.7 GB

60 seconds

0

# #1 Really bad vs good



## #2 AsNoTracking

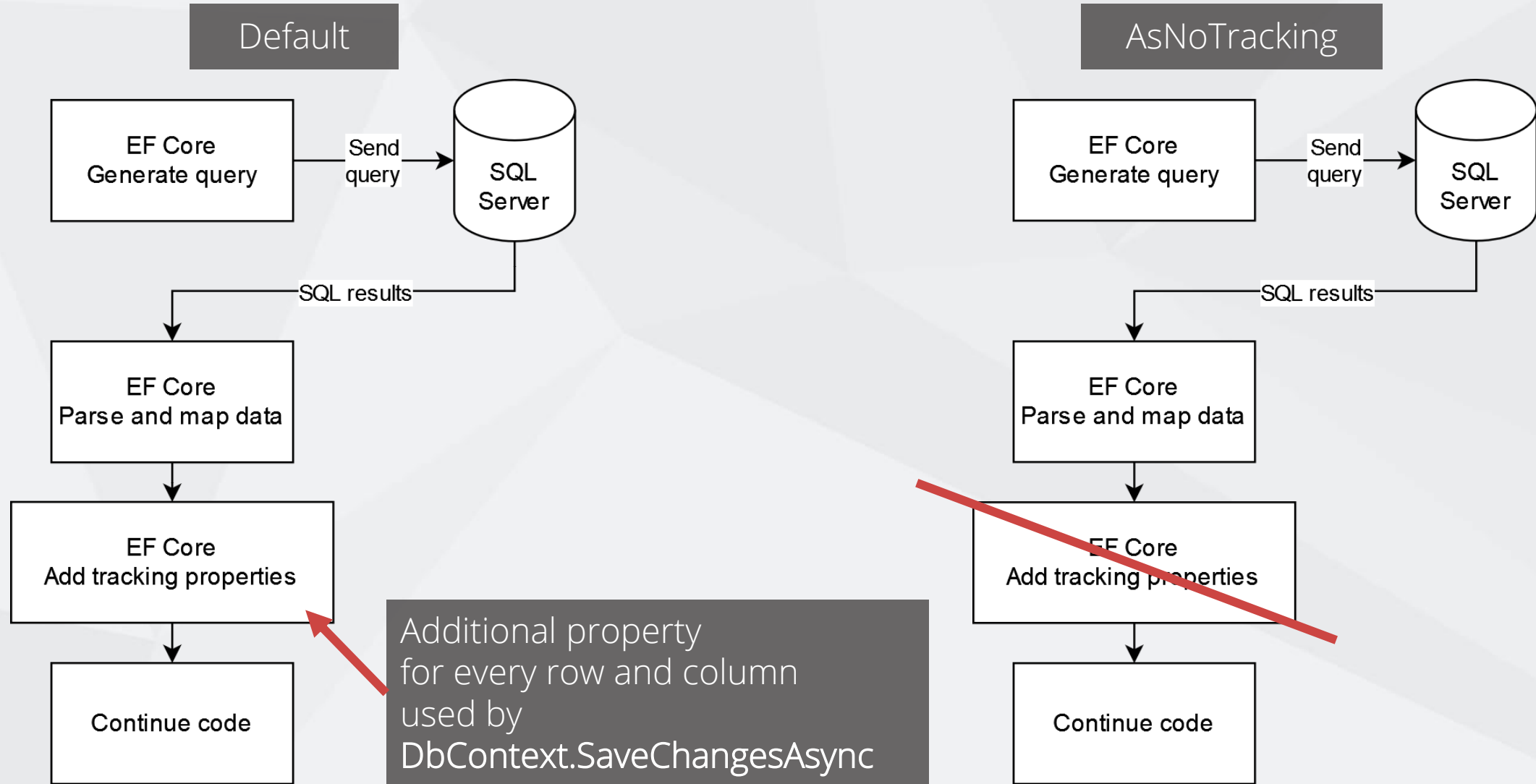
Exclude change tracking on entities

Less memory and CPU usage

⚠ Changes to the entities are **not tracked** ⚠

```
[Benchmark]
0 references | 0 changes | 0 authors, 0 changes
public List<Sale> GetSalesAsNoTracking()
{
    return SalesDbContext.Sales
        .AsNoTracking()
        .Where(x => x.CustomerId == 7)
        .ToList();
}
```

## #2 What is happening?





## #2 What is the impact?

Method	Mean	Error	StdDev	Ratio	Gen 0	Gen 1	Gen 2	Allocated
GetSalesQueryable	2,014.2 ms	39.37 ms	36.83 ms	1.00	65000.0000	19000.0000	2000.0000	582 MB
GetSalesAsNoTracking	387.4 ms	7.64 ms	13.58 ms	0.19	17000.0000	7000.0000	-	147 MB

~1.6 seconds faster!

~5 times faster

~4x less memory

⚠ Results be wildly different between queries and DB providers! ⚠  
Some found it has no effect on SQLite DB provider

## #2 What about load testing?

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesWhere/where
Bombarding https://localhost:5001/ExamplesWhere/where for 1m0s using 5 connection(s)
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       3.68      12.10     199.96
Latency        1.40s     371.46ms    2.66s
```

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesWhere/whereNoTracking
Bombarding https://localhost:5001/ExamplesWhere/whereNoTracking for 1m0s using 5 connection(s)
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec      14.72      13.34     249.94
Latency       358.92ms    105.40ms    1.48s
```

~4x faster on average in a load test!

## #3 Explicit includes

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


They are always included even if not needed

Devs often forget to remove them

We get all columns even if we don't need them

# #3 & #4 Implicit includes with Select

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



```
var query = _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.SalesPerson.LastName
    });
```

# #3 & #4 Load Test

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesJoin/badCase
Bombarding https://localhost:5001/ExamplesJoin/badCase for 1m0s using 5 connection(s)
```

```
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec        2.90       11.85     238.05
Latency         1.92s     561.85ms     5.16s
```

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesJoin/implicitJoin
Bombarding https://localhost:5001/ExamplesJoin/implicitJoin for 1m0s using 5 connection(s)
```

```
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec        7.49       25.53     263.10
Latency        748.18ms     575.21ms     5.30s
```

~2.5x faster on average in a load test!

# #5 Common Pagination issues

- Inefficient code
  - Filter data
  - Get **all** filtered data
  - Count them
  - Apply Take and Skip **in-memory** for pagination



## #5 Pagination code – naïve

```
IQueryable<Sale> query = _dbContext.Sales
    .AsNoTracking()
    // Very commonly used in combination of in-memory pagination, making things even worse
    .Include(x => x.SalesPerson)
    .Where(x => x.SalesPersonId == salesPersonId);

List<Sale> dbResult = await query.ToListAsync(ct);
int count = dbResult.Count;
List<SalesWithSalesPerson> result = dbResult
    .Skip(page * pageSize)
    .Take(pageSize)
    .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.SalesPerson.LastName
    })
    .ToList();
```

Gets all filtered data

Count in-memory

Pagination is done

In-memory

# #5 Pagination code – improved

```
IQueryable<SalesWithSalesPerson> query = _dbContext.Sales
    .AsNoTracking()
    .Where(x => x.SalesPersonId == salesPersonId)
    .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.SalesPerson.LastName
    });
```

We are preparing the query  
For counting and pagination

```
// After all conditions are applied, count them in DB.
```

```
int count = await query.CountAsync(ct)
```

Count in SQL Server

```
// Apply paginations, sorts and more complex select statements.
```

```
query = query
```

```
.Skip(page * pageSize)
.Take(pageSize);
```

Pagination is done  
on SQL Server

```
List<SalesWithSalesPerson> result = await query.ToListAsync(ct)
```

# #5 Load test

```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesPaginations/worstCase
Bombarding https://localhost:5001/ExamplesPaginations/worstCase for 1m0s using 5 connection(s)
[=====
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       3.72      17.21     210.48
Latency        1.35s     365.60ms    2.27s
```

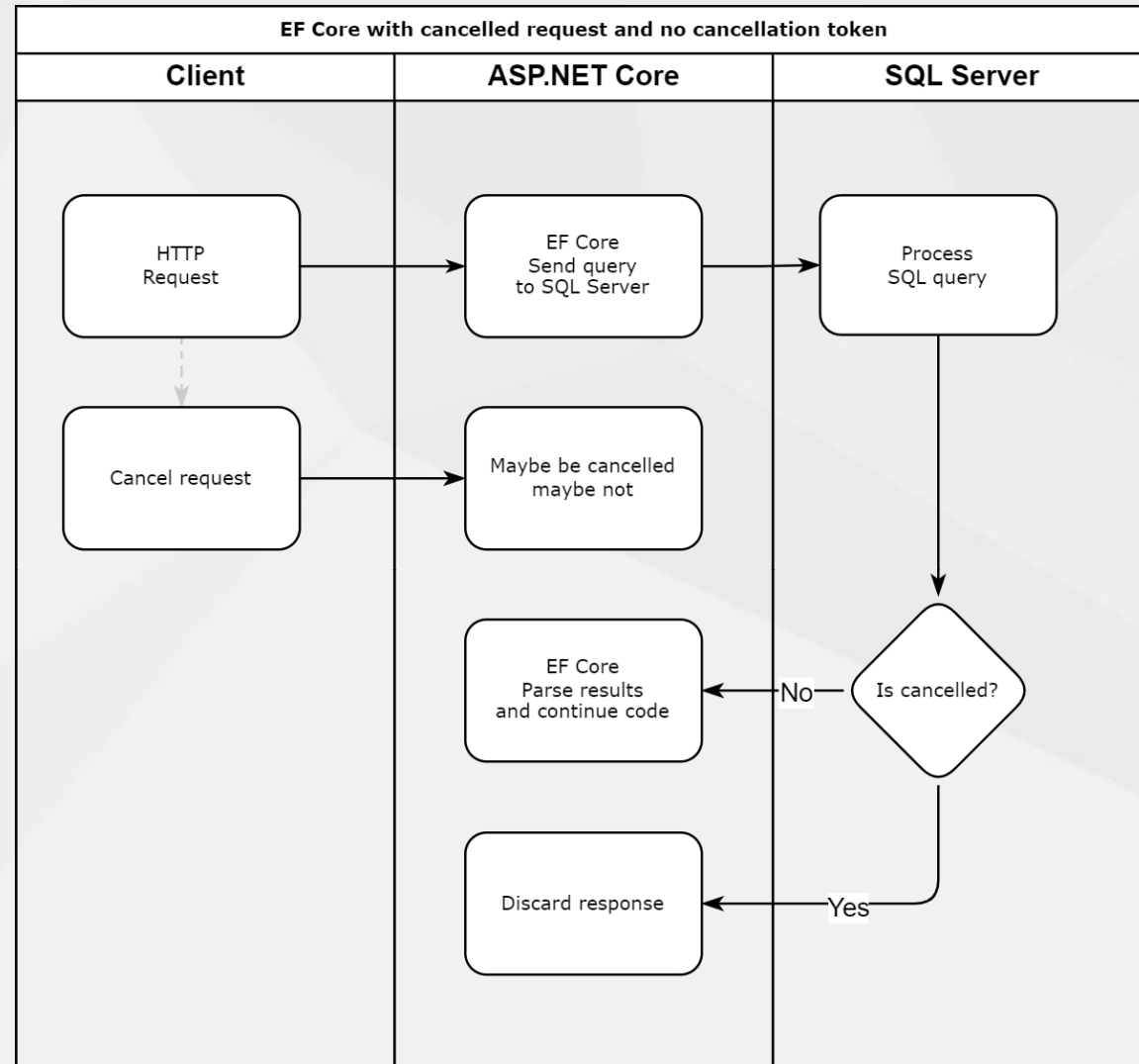
```
> bombardier -c 5 -t 60s -d 60s -l https://localhost:5001/ExamplesPaginations/executedOnDB
Bombarding https://localhost:5001/ExamplesPaginations/executedOnDB for 1m0s using 5 connection(s)
[=====
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       65.41      21.98     263.10
Latency        76.94ms    13.79ms    237.05ms
```

~18x faster on average in a load test!

# #6 Non-cancellable queries

- When query is running on SQL Server
  - It will run until completion
- Complex queries can stuff up SQL Server

# #6 What's the problem?




# #6 Async and CancellationToken

Stop execution on the server when cancelled

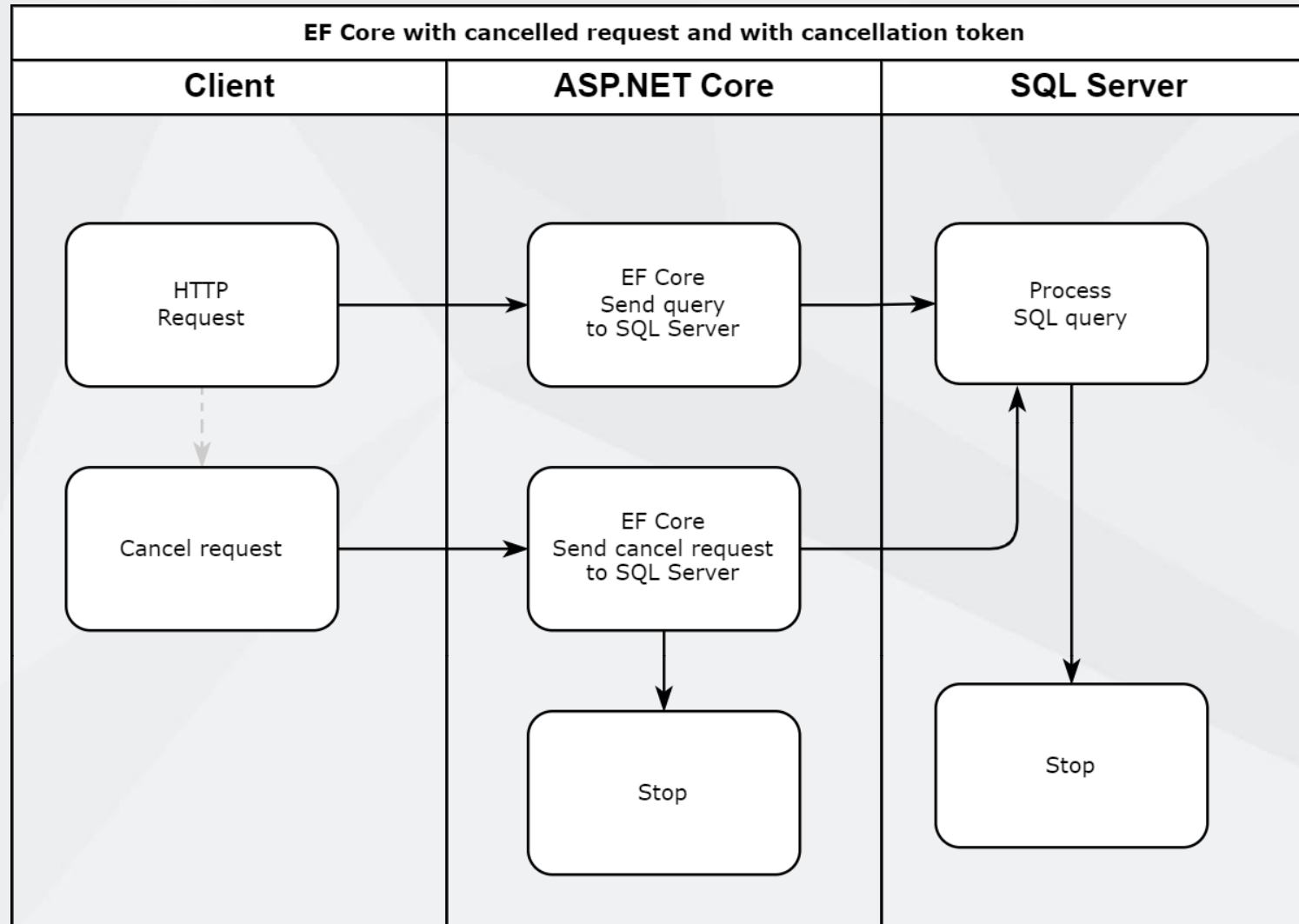
```
[HttpGet("splitQuery")]
0 references | Jernej Kavka (JK), 65 days ago | 1 author, 1 change
public async Task<TestResult<int>> SplitQuery(CancellationToken ct)
{
    var query = _dbContext.Sales
        .AsNoTracking()
        .AsSplitQuery()
        .Where(x => x.SalesPersonId == 1)
        .Select(x => new SalesWithSalesPerson
        {
            CustomerId = x.CustomerId,
            SalesId = x.SalesPersonId,
            ProductId = x.ProductId,
            Quantity = x.Quantity,
            SalesPersonId = x.SalesPersonId,
            SalesPersonFirstName = x.SalesPerson.FirstName,
            SalesPersonLastName = x.SalesPerson.LastName
        });

    List<SalesWithSalesPerson> result = await query.TagWithContext().ToListAsync(ct);
}
```





# #6 How it works cancellation token?



# #6 Load test

```
> bombardier -c 12 -t 1s -d 60s -l https://localhost:5001/test/count
Bombarding https://localhost:5001/test/count for 1m0s using 12 connection(s)
[=====
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       1.64      28.10     666.49
Latency        10.14s     2.83s     17.89s
```

```
> bombardier -c 12 -t 1s -d 60s -l https://localhost:5001/test/countasync-ct
Bombarding https://localhost:5001/test/countasync-ct for 1m0s using 12 connection(s)
[=====
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       2.04      32.15     631.45
Latency         5.85s    368.77ms     7.26s
```

~2x faster on average in a load test!  
Also, responsiveness returns 40 seconds faster

# #7 Inefficient update/delete #1

- Before EF7 we had to fetch data before could update
- Entities need to be tracking
- Can be very inefficient

# #7 Inefficient update/delete #2

```
IQueryable<Employee> query = GetBaseQuery(isLoadFriendly);
var employees = query.ToList();
foreach (var employee in employees)
{
    string firstName = employee.FirstName;
    employee.FirstName = employee.LastName;
    employee.LastName = firstName;
}

_dbContext.SaveChangesAsync();
```

```
-- Get all employees that we want to swap first and last name.
-- We need to fetch all columns as it needs to be tracking!
```

```
SELECT TOP(@__p_0) [e].[EmployeeID], [e].[FirstName], [e].[LastName], [e].[MiddleInitial]
FROM [Employees] AS [e]
```

```
SET NOCOUNT ON;
UPDATE [Employees] SET [FirstName] = @p0, [LastName] = @p1
OUTPUT 1
WHERE [EmployeeID] = @p2;
```

```
-- Lots of updates ...
UPDATE [Employees] SET [FirstName] = @p63, [LastName] = @p64
OUTPUT 1
WHERE [EmployeeID] = @p65;
UPDATE [Employees] SET [FirstName] = @p66, [LastName] = @p67
OUTPUT 1
WHERE [EmployeeID] = @p68;
```

# #7 Inefficient update/delete #3

```
IQueryable<Employee> query = GetBaseQuery(isLoadFriendly);  
var employees = query  
    .TagWith("Swap first and last name")  
    .ExecuteUpdate(x => x  
        .SetProperty(p => p.FirstName, b => b.LastName)  
        .SetProperty(p => p.LastName, b => b.FirstName));  
  
_dbContext.SaveChangesAsync();
```

```
-- Swap first and last name
```

```
UPDATE [e]  
SET [e].[LastName] = [e].[FirstName],  
    [e].[FirstName] = [e].[LastName]  
FROM [Employees] AS [e]
```

# #7 Inefficient update/delete #4

```
~#> bombardier -c 12 -t 1s -d 60s -l https://localhost:5001/ExamplesUpdate/worstCase
Bombarding https://localhost:5001/ExamplesUpdate/worstCase for 1m0s using 12 connection(s)
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec      524.96    478.57    3168.91
Latency       22.87ms   54.64ms   1.28s
```

```
~#> bombardier -c 12 -t 1s -d 60s -l https://localhost:5001/ExamplesUpdate/updateQuery
Bombarding https://localhost:5001/ExamplesUpdate/updateQuery for 1m0s using 12 connection(s)
[=====]
Done!
Statistics      Avg      Stdev      Max
Reqs/sec     1659.20    479.76    2781.09
Latency       7.23ms    26.63ms   1.66s
```

For only 23 records, it's still ~3x faster!  
Performance improvements can be massive depending on the query!



# Mini Best Practices

# Bonus #1 - Other things I have seen

0 references | 0 changes | 0 authors, 0 changes

```
public bool HasAny()
{
    var employees = _dbContext.Employees
        .Where(x => x.EmployeeId == 1)
        .ToList();
    return employees.Count > 0;
}
```

0 references | 0 changes | 0 authors, 0 changes

```
public bool HasAny()
{
    return _dbContext.Employees
        .Any(x => x.EmployeeId == 1);
}
```

0 references | 0 changes | 0 authors, 0 changes

```
public List<SaleModel> GetSales(int? saleId)
{
    var sales = _dbContext.Sales
        .Include(x => x.Product)
        .ToList();

    if (saleId != null)
    {
        sales = sales
            .Where(x => x.SalesId == saleId.Value)
            .ToList();
    }

    return sales.Select(x => new SaleModel
    {
        SaleId = x.SalesId,
        Quantity = x.Quantity,
        ProductName = x.Product.Name
    }).ToList();
}
```

0 references | 0 changes | 0 authors, 0 changes

```
public List<SaleModel> GetSales(int? saleId)
{
    IQueryable<Sale> salesQuery = _dbContext.Sales;

    if (saleId != null)
    {
        salesQuery = salesQuery
            .Where(x => x.SalesId == saleId.Value);
    }

    return salesQuery.Select(x => new SaleModel
    {
        SaleId = x.SalesId,
        Quantity = x.Quantity,
        ProductName = x.Product.Name
    }).ToList();
}
```

# Bonus #2

## DbContextPool

- Reuses existing **DbContext**
- Fewer connections to SQL Server
- Improves resiliency

```
services.AddDbContextPool<SalesDbContext>(options =>  
{  
    options.UseSqlServer(Configuration.GetConnectionString("SalesDB"));  
});
```

# Bonus #3

## TagWith

Adds comments into SQL query for easier debugging

```
var list = await _context.Tweets
    .TagWith("GetTweets")
    .ToListAsync(cancellationToken)
    .ConfigureAwait(false);
```



```
-- GetTweets
```

```
SELECT [t].[Id], [t].[CreatedUtc], [t].[Message], [t].[Username]
FROM [Tweets] AS [t]
```

# Bonus #4 – ChatGPT

ChatGPT **PLUS**

You're an EF Core expert and clean code developer.

<user's instructions>

DbContext:

<Optional project's DbContext, can be trimmed to include only the relevant parts>

Entities:

<Only necessary entities>

Return types:

<Optional return type we want to map>



ChatGPT may produce inaccurate information about people, places, or facts. [ChatGPT May 12 Version](#)

Join the Conversation @SSW\_TV @jernej\_kavka #EFCore

# Bonus #4 – ChatGPT

🌟 Model: GPT-4



You're an EF Core expert and clean code developer.

Get all employees details, for each get statistics on how many products they earned and a list of sales they did, all paginated.

DbContext:

```
public partial class SalesDbContext : DbContext
{
    public SalesDbContext(DbContextOptions<SalesDbContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Customer> Customers { get; set; }
```

```
public async Task<List<EmployerStats>> GetEmployeeStats(int pageNumber,
{
    return await _context.Employees
        .OrderBy(e => e.EmployeeId)
        .Skip((pageNumber - 1) * pageSize)
        .Take(pageSize)
        .Select(e => new EmployerStats
        {
            FirstName = e.FirstName,
            LastName = e.LastName,
            TotalProductSold = e.Sales.Count,
            TotalSales = e.Sales.Sum(s => s.Quantity * s.Product.Price),
            TotalQuantity = e.Sales.Sum(s => s.Quantity),
            Sales = e.Sales.Select(s => new SaleModel
            {
                SaleId = s.SalesId,
                ProductName = s.Product.Name,
                Quantity = s.Quantity
            })
        })
        .ToListAsync();
}
```

# Before we end #1

When is something **not** premature optimization?

- ✓ Following best practices for given scenario

- ✓ When high-load or high-data is expected

  - i Talk to Product Owner and make it part of acceptance criteria

- ✓ When it already causes performance issues

  - i Talk to Product Owner and make it part of acceptance criteria or create tech debt PBI/issue and mention it in the code

  - [ssw.com.au/rules/technical-debt/](https://ssw.com.au/rules/technical-debt/)



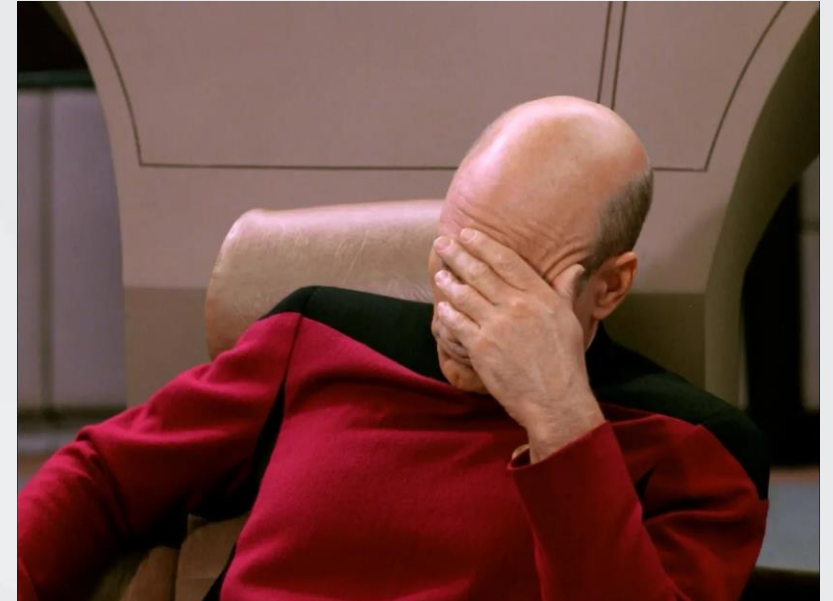
# Before we end #2

## What should you not do?

### ❌ “We’ll fix it later” with no follow-up

ℹ️ Create a tech debt PBI/issue and a link to it in code as a TODO comment

🌐 [ssw.com.au/rules/technical-debt/](https://ssw.com.au/rules/technical-debt/)



### ❌ “Premature optimization” as an excuse for bad code

ℹ️ There are valid reason to not optimize your code, bad code is not one of them!



```
// TODO: [Tech-Debt] PBI-52785: ⚡ ✨ Admin Azure DevOps | Fetching Azure DevOps settings pull 17k+ timesheets
```

# Resources



Source code for examples and benchmarking

[github.com/jernejk/EFCoreSamples.StabilityAndPerformance](https://github.com/jernejk/EFCoreSamples.StabilityAndPerformance)

Microsoft resources for better EF Core performance

[docs.microsoft.com/en-us/ef/core/performance/](https://docs.microsoft.com/en-us/ef/core/performance/)

Rules to Better Entity Framework

[ssw.com.au/rules/rules-to-better-entity-framework](https://ssw.com.au/rules/rules-to-better-entity-framework)

Rules to Better LINQ

[ssw.com.au/rules/rules-to-better-linq](https://ssw.com.au/rules/rules-to-better-linq)

Nick Chapsas YouTube – Great breakdowns of features and best practices

[youtube.com/watch?v=Q4LtKa\\_HTHU](https://youtube.com/watch?v=Q4LtKa_HTHU)

Join the Conversation @SSW\_TV @jernejkavka #EFCore

# Key takeaways 🗝️

- **IEnumerable** is dangerous when used with EF Core
- **AsNoTracking** has massive performance impact
  - Make sure you don't need to update/delete the entities
- Simpler code can result in better performance

# The sequel – NDC Sydney 2023

**NDC** { Sydney }

**Thursday** Room 4 17:40 - 18:40  
(UTC+11) Talk (60 min)

**Unlocking the Full  
Potential: The  
Harmonious Dance of  
EF Core and SQL Server**

Find this presentation on GitHub!



[github.com/sswconsulting/presentations](https://github.com/sswconsulting/presentations)

The background is a light gray with a subtle geometric pattern of overlapping triangles. It is decorated with festive elements: red and blue streamers, red and blue triangles of various sizes, and red and blue fireworks exploding on the right side. In the bottom left corner, there is a small grid of squares in shades of gray and red.

Thank you!

[info@ssw.com.au](mailto:info@ssw.com.au)

[www.ssw.com.au](http://www.ssw.com.au)

Sydney | Melbourne | Brisbane

# Questions?

[info@ssw.com.au](mailto:info@ssw.com.au)

[www.ssw.com.au](http://www.ssw.com.au)

Sydney | Melbourne | Brisbane

