Rediscovering Entity Framework Core: Speed, Flexibility, and Power for Modern Development

Barret Blake

Barret Blake

- Husband
- Father
- Microsoft MVP – Power Automate
- Azure Application Architect
- Speaker
- Blogger
- Developer
- Model Railroader
- Gamer
- Buckeyes Fan

MVP Microsoft® Most Valuable Professional

EF Historical Problems

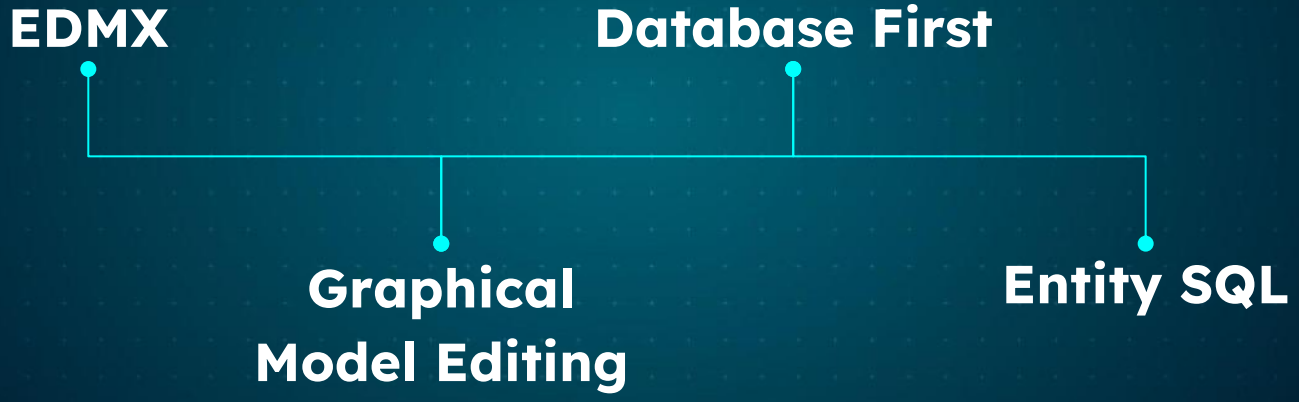
Performance	Complexity	Control	Other
• Overhead • Inefficient SQL • Cold Queries • N+1 Queries	• Learning Curve • Migration • EDMX	• Lack of Control • Hidden Behaviors	• Not Thread Safe • Code-First • Versioning

2016

Entity Framework Core

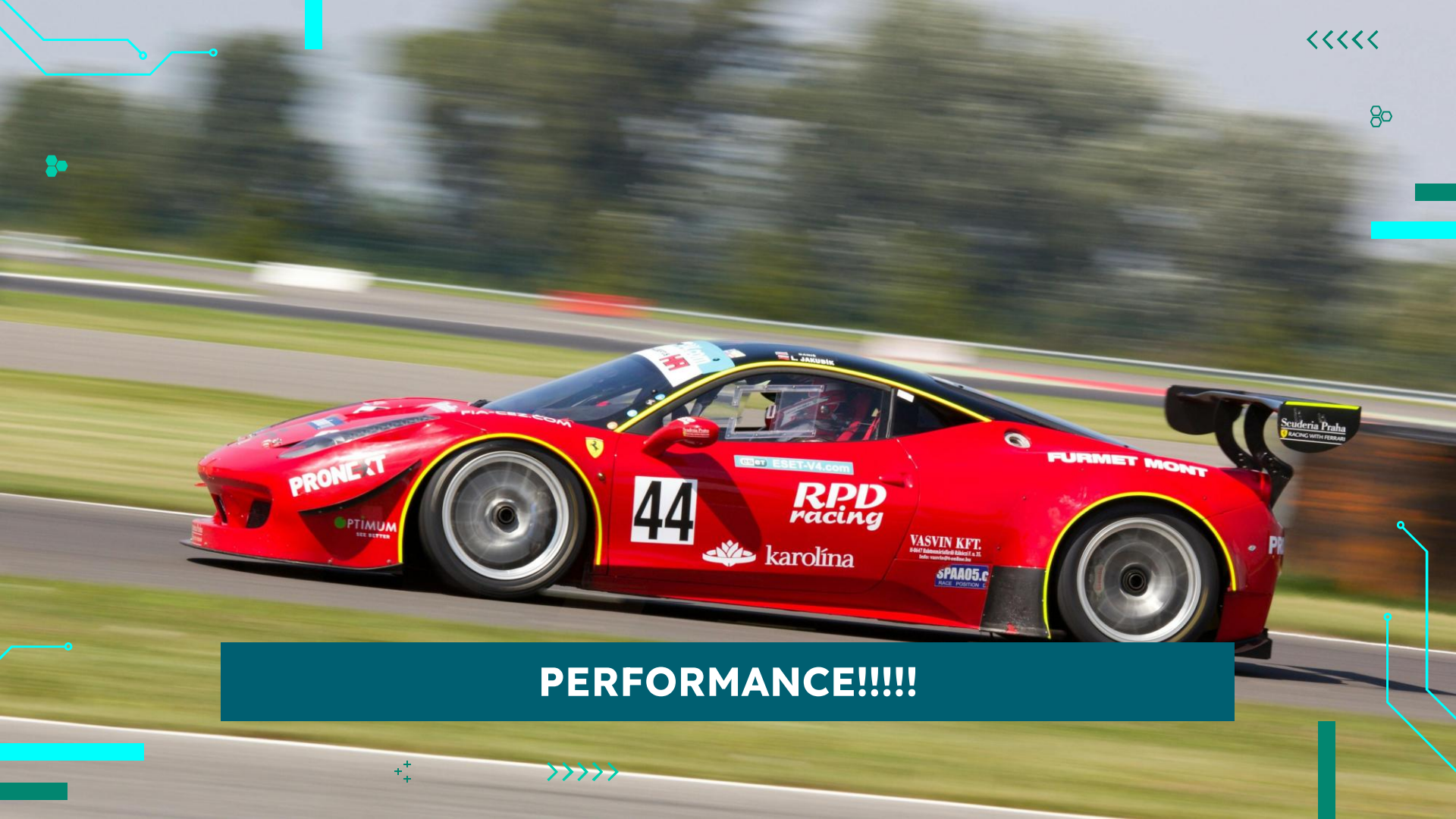
v1

What's Missing?



What's New?

- Cross platform – Linux/Mac/Windows
- Property value conversions
- Constructors w/parameters
- Alternate keys
- Shadow state properties
- Client key generation
- Global query filters
- Mapped types w/no keys
- Field mapping
- Nullable reference types
- Eager loading for derived types
- Raw SQL support for LINQ
- Await foreach
- Statement batching
- DbContext pooling
- Support for Jet, CosmosDB, in-memory



PERFORMANCE!!!!!

vs .NET EF 6

First or Default

	Mean	Error	StdDev	Median
EF 6	809.4 us	83.65 us	235.94 us	702.7 us
EF Core 3.1	466.5 us	19.20 us	54.48 us	467.5 us

Fetching 10,000 Records

	Mean	Error	StdDev	Median
EF 6	183.29 ms	3.898 ms	3.828 ms	181.72 ms
EF Core 3.1	81.31 ms	1.510 ms	1.438 ms	81.46 ms

src: https://chadgolden.com/blog/comparing-performance-of-ef6-to-ef-core-3

vs .NET EF 6

Add

	Mean	Error	StdDev	Median
EF 6	1,850.6 us	157.26 us	295.38 us	1,746.3 us
EF Core 3.1	1,411.7 us	76.51 us	147.40 us	1,391.8 us

Update

	Mean	Error	StdDev	Median
EF 6	949.3 us	72.86 us	135.05 us	899.5 us
EF Core 3.1	663.2 us	51.18 us	96.14 us	653.1 us

Delete

	Mean	Error	StdDev	Median
EF 6	2,849.7 us	194.78 us	370.59 us	2,768.4 us
EF Core 3.1	2,447.7 us	163.57 us	319.03 us	2,321.5 us

src: https://chadgolden.com/blog/comparing-performance-of-ef6-to-ef-core-3

vs Dapper

Single record

	Mean	Error	StdDev	Allocated
Dapper_GetById_FirstAsync	1.166 ms	0.0224 ms	0.0230 ms	13.15 KB
Dapper_GetById_FirstOrDefaultAsync	1.174 ms	0.0212 ms	0.0297 ms	13.15 KB
Dapper_GetById_SingleAsync	1.137 ms	0.0146 ms	0.0136 ms	13.25 KB
EfCore_GetById_By_Qry	1.213 ms	0.0241 ms	0.0226 ms	28.55 KB
EfCore_GetById_By_FirstAsync	1.200 ms	0.0176 ms	0.0156 ms	19.97 KB
EfCore_GetById_By_FirstOrDefaultAsync	1.219 ms	0.0237 ms	0.0273 ms	19.98 KB
EfCore_GetById_By_SingleAsync	3.543 ms	0.0444 ms	0.0415 ms	21.14 KB

src: https://trailheadtechnology.com/ef-core-9-vs-dapper-performance-face-off/

vs Dapper

ToList() (14,000 items)

	Mean	Error	StdDev	Allocated
EFCore_To_List_Raw	5.861 ms	0.0777 ms	0.0727 ms	930.7 KB
EFCore_To_List_LINQ	5.862 ms	0.1122 ms	0.2214 ms	927.56 KB
Dapper_To_List	5.643 ms	0.0615 ms	0.0575 ms	1460.89 KB

src: https://trailheadtechnology.com/ef-core-9-vs-dapper-performance-face-off/

vs Dapper

Insert Single & Insert Range (30 records)

	Mean	Error	StdDev	Allocated
Dapper_Insert_One_Async	18.27 ms	0.320 ms	0.300 ms	18.23 KB
EF_Core_Insert_One_Async	17.91 ms	0.234 ms	0.195 ms	39.09 KB
Dapper_Insert_Range_Async	22.96 ms	0.437 ms	0.569 ms	427.73 KB
EF_Core_Insert_Range_Async	22.58 ms	0.201 ms	0.178 ms	753.61 KB

src: https://trailheadtechnology.com/ef-core-9-vs-dapper-performance-face-off/

vs Dapper

Update

	Mean	Error	StdDev	Allocated
Dapper_Update_Single_Async	169.2 us	3.24 us	4.10 us	3.68 KB
EFCore_Update_Single_Async	209.1 us	4.07 us	3.81 us	61.33 KB

src: https://trailheadtechnology.com/ef-core-9-vs-dapper-performance-face-off/

Which When?

EF Core	Dapper
• ORM • Features • Linq Query Support built-in • Database versioning	• Super high-performance scenarios • Massive data-sets

Cross-Platform

Windows

Linux

Mac

Property Value Conversions

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    //Property Value Converter
    modelBuilder.Entity<Character>()
        .Property(c => c.Rank)
        .HasConversion(
            a => a.ToString(),
            a => (Enums.Rank)Enum.Parse(typeof(Enums.Rank), a));
}
```

Entity Constructors

```
public class Character:Entity
{
    public Character() { }  //public constructor

    //EF Core will call this constructor
    private Character(string firstName, string lastName, Enums.Races race)
    {
        FirstName = firstName;
        LastName = lastName;
        Race = race;
    }

    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Enums.Races Race { get; set; }
    public Enums.Rank Rank { get; set; }
}
```

Keyless Entity Types

```
[Keyless]
public class CharacterEpisode
{
    public string CharacterName { get; set; }
    public string EpisodeName { get; set; }
}
```

Alternate Keys

```
public class Season
{
    public int Id { get; set; }
    public int SeasonNumber { get; set; }
    public List<Episode> Episodes { get; set; };
}

public class Episode
{
    public int Id { get; set; }
    public int EpisodeNumber { get; set; }
    public string Title { get; set; } = null!;
    public int SeasonId { get; set; }
    public Season Season { get; set; } = null!;
}
```

```
modelBuilder.Entity<Episode>()
    .HasOne(s => s.Season)
    .WithMany(e => e.Episodes)
    .HasForeignKey(s => s.SeasonId)
    .HasPrincipalKey(e => e.SeasonNumber);
```

Shadow State Properties

```
modelBuilder.Entity<Character>()
    .Property<DateTime>("LastUpdated");
```

```
var sinclair = db.Characters
    .FirstOrDefault(c => c.FirstName == "Jeffrey");

db.Entry(sinclair).Property("LastUpdated").CurrentValue = DateTime.Now;

var characters = db.Characters
    .OrderBy(x=>EF.Property<DateTime>(x, "LastUpdated"));
```

Client Generated Keys & Computed

```
[Key]
public int Id { get; set; }
```

```
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]
public DateTime LastUpdated { get; set; }
```

Global Query Filters

```
public SciFiContext(DbContextOptions<SciFiContext> opts, IUniverseService universe)
{
    _universe = universe;
}
```

```
public class Character:Entity
{
    public string Universe { get; set; }
    ...
}
```

```
modelBuilder.Entity<Character>()
    .HasQueryFilter(ch => ch.Universe == _universe.Universe);
```

Global Query Filters

```
modelBuilder.Entity<Character>()
    .HasQueryFilter(ch => ch.Universe == _universe.Universe && !ch.IsDeleted);
```

```
var allCharacters = db.Characters
    .OrderBy(x=>x.Universe).ThenBy(x=>x.LastName).ThenBy(x=>x.FirstName)
    .IgnoreQueryFilters()
    .ToList();
```

Raw SQL

```
var getCharacters = db.Characters
    .FromSql($"SELECT * FROM Characters")
    .ToList();
```

```
var universeName = "Babylon 5";
var b5Characters = db.Characters
    .FromSql($"EXECUTE dbo.FetchCharactersByUniverse {universeName}")
    .ToList();
```

```
var columnName = "Rank";
var columnValue = new SqlParameter("columnValue", "Commander");
var justCaptains = db.Characters
    .FromSqlRaw($"SELECT * FROM Characters WHERE {columnName} = @columnValue", columnValue)
    .ToList();
```

Raw SQL

```
var getB5Characters = db.Characters
    .FromSql($"SELECT * FROM Characters")
    .Where(ch=>ch.Universe == "Babylon 5")
    .Include(ch=>ch.Episodes)
    .ToList();
```

Statement Batching

```
foreach (var character in db.Characters)
{
    character.Race = Enums.Races.Human;
}
db.SaveChanges();
```

```
db.Characters.ExecuteUpdate(ch=> ch.SetProperty(c=>c.Race, Enums.Races.Human));
```

DbContext Pooling

```
builder.Services.AddDbContextPool<SciFiContext>(
    o => o.UseSqlServer(builder.Configuration.GetConnectionString("SciFiDbConnection")));
```

Complex Types

```
public class Customer {
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Address Address { get; set; }
}

public Class Address {
    public string Street { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Zip { get; set; }
}
```

Complex Types

```
public class Customer
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Address Address { get; set; }
}

[ComplexType]
public class Address
{
    public string Street { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Zip { get; set; }
}
```

Complex Types

Column Name	Data Type	Allow Nulls
Id	uniqueidentifier	☐
FirstName	nvarchar(MAX)	☐
LastName	nvarchar(MAX)	☐
Address_City	nvarchar(MAX)	☐
Address_State	nvarchar(MAX)	☐
Address_Street	nvarchar(MAX)	☐
Address_Zip	nvarchar(MAX)	☐
		☐

Primitive Collections

```
public class Customer
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Address Address { get; set; }
    public List<Guid> WishListProducts { get; set; }
}
```

Primitive Collections

Column Name	Data Type	Allow Nulls
Id	uniqueidentifier	☐
FirstName	nvarchar(MAX)	☐
LastName	nvarchar(MAX)	☐
WishListProducts	nvarchar(MAX)	☐
Address_City	nvarchar(MAX)	☐
Address_State	nvarchar(MAX)	☐
Address_Street	nvarchar(MAX)	☐
Address_Zip	nvarchar(MAX)	☐
Address_Coordinates_Latitude	float	☐
Address_Coordinates_Longitude	float	☐

```
[
 "ad39b7c2-9011-41b7-b2ca-716e627527d1",
 "97137edf-b7f4-4db3-b9ae-76a51c4a3b48",
 "78ab4609-b58c-4107-8b5c-c2b0fecc39df",
 "00b3c576-dd34-4905-ba99-f37cef1f327d"
]
```

Primitive Collections

```
Guid productId = new Guid("f5b3f4b3-3b4b-4b4b-8b4b-4b4b4b4b4b4b");
var customersWhoWantItem = await _context.Customers
    .Where(x ⇒ x.WishListProducts.Contains(productId))
    .ToListAsync();
```

```
var firstNames = new[] { "John", "Jane", "Jim", "Jill" };
var customers = await _context.Customers
    .Where(c ⇒ firstNames.Contains(c.FirstName))
    .ToListAsync();
```

Compiled Queries

	# Records	Mean	Error	StdDev	Allocated
WithCompiledQuery	1	564.2 us	6.75 us	5.99 us	9 KB
WithoutCompiledQuery	1	671.6 us	12.72 us	16.54 us	13 KB
WithCompiledQuery	10	645.3 us	10.00 us	9.35 us	13 KB
WithoutCompiledQuery	10	709.8 us	25.20 us	73.10 us	18 KB

```
private static readonly Func<BloggingContext, int, IAsyncEnumerable<Blog>> _compiledQuery
    = EF.CompileAsyncQuery(
        (BloggingContext context, int length) => context.Blogs.Where(b => b.Url.StartsWith("http://") &&
b.Url.Length == length));
```

src: https://learn.Microsoft.com/en-us/ef/core/performance/advanced-performance-topics

Other Features

Field Mapping	Allows you to support scenarios such as INotifyPropertyChanged without triggering the IsDirty property
Nullable Reference Types	Full support for the « new » C# nullable reference types
Eager Loading for Derived Types	You can now use .Include and .ThenInclude for eager loading of derived types as well as other related data
Await ForEach	Full support for efficient async Linq queries in ForEach loops
More Data Sources	Cosmos DB, In-Memory, Jet (Microsoft Access)
JSON	Full support for OpenJSON, JSON data fields, and other document related stuff

Wrapping It Up

Thank you!

Do you have any questions?
https://linkedin.com/in/barretblake
https://barretblake.dev
 @barretblake.dev