



NHibernate – eine Alternative zu EF Core?

Olaf Lischke

Lischke EDV / www.it-visions.de

Speaker: Olaf Lischke



kontaktdaten.vcf



macht .NET, seit es .NET gibt



versucht, Projekte und Seminare zu kon



singt Tenor in Chören und Musikprojekten



zockte schon auf dem ZX 81, heute ausschließlich auf PC



fotografiert, seit er eine Kamera halten kann



fliegt, wenn Wetter und Zeit es zulassen (TMG/SEP)

O/R-Mapper im Vergleich: Herkunft

NHibernate

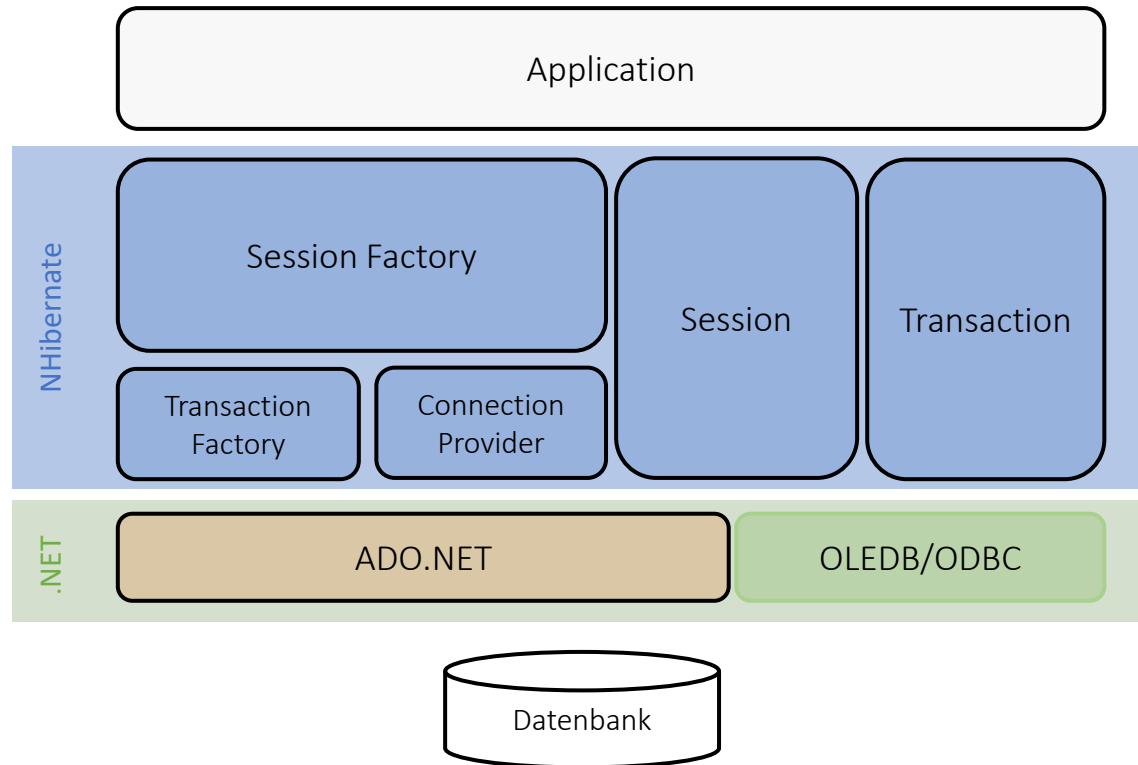
- 2003 Portierung von Java
- Seit 2006 komplett Open Source
- Vollständig in C# und .NET entwickelt:
 - Unterstützt eine Vielzahl von DB-Systemen
 - lebhafte Community mit div. Unter- und Zusatzprojekten
- Verfügbar als nuget-Paket:
 - Aktuelle Version 5.5.3 (August 2025)
 - .NET 6
 - .NET Standard 2.0
 - .NET Framework 4.6.1
 - Lizenz LGPL-2.1-only

Entity Framework/EF Core

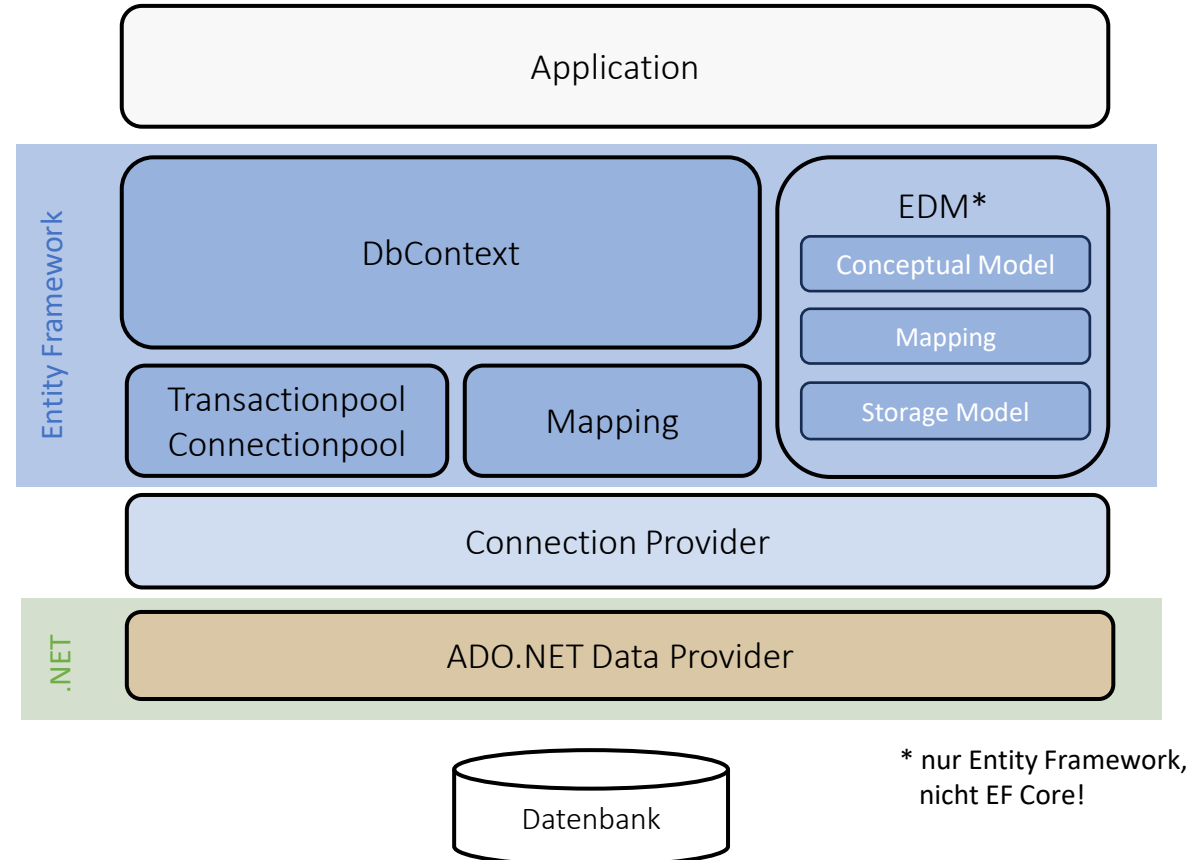
- 2008 von Microsoft als Bestandteil von .NET Framework 3.5 SP1
 - damals (bis .NET Framework 4.8) bevorzugt SQL-Server, jedoch einige Treiber von Drittanbietern (z.B. Oracle)
- 2016 Neuentwicklung EF Core, seitdem:
 - quelloffen
 - nuget-Paket(e)
 - Unterstützung vieler DB-Systeme
 - MIT Lizenz
- Seit EF Core 6: **PostgreSQL** primäre Entwicklungsplattform

O/R-Mapper im Vergleich: Architektur I

NHibernate



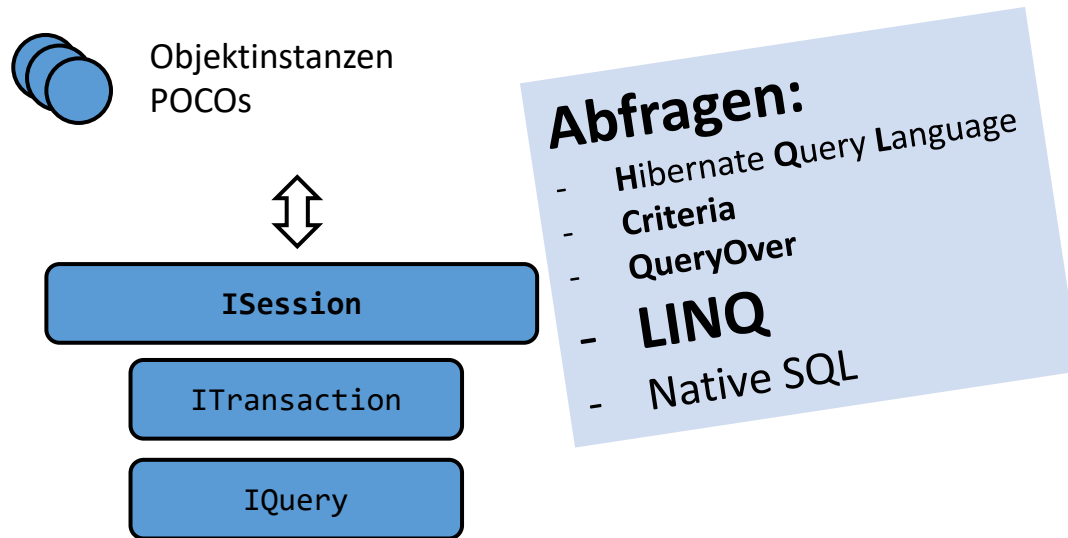
Entity Framework/EF Core



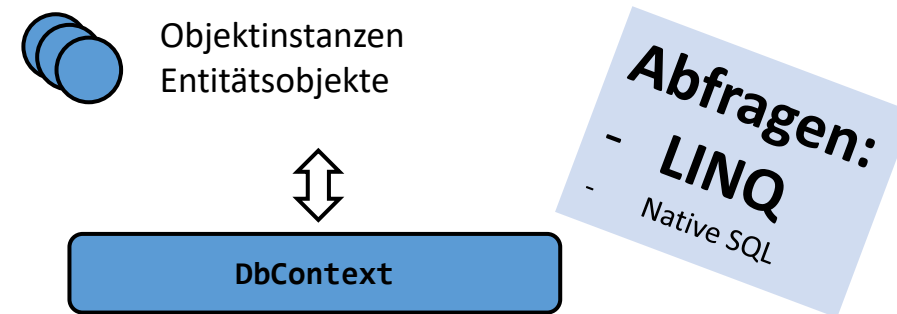
* nur Entity Framework, nicht EF Core!

O/R-Mapper im Vergleich: Architektur II

NHibernate



Entity Framework/EF Core



Code-First/Reverse-Engineering*

* Reverse-Engineering: formerly know as DataBase-First

NHibernate

- Code-First:
 - SchemaExport/SchemaUpdate
 - div. Drittanbieter-Tools
- Reverse Engineering
 - manuell
 - Drittanbieter-Tools
 - z.B. DevArt Entity Developer
(kann auch Entity Framework/EF Core)

Entity Framework/EF Core

- Entity Framework:
 - Grafischer Designer für EDMX
- EF Core:
 - „Echtes“ Code-First inkl. Migrations und Schema-Anpassungen per Powershell
 - Reverse-Engineering per Powershell
 - Beides mit Hilfe von `Microsoft.EntityFrameworkCore.Tools`

Konfiguration und Mapping

NHibernate

- POCO-Klassen
 - alle gemappten Properties `virtual`
 - nicht gemappte Properties benötigen keine Konfiguration
- Mapping
 - HBML
 - FluentApi (Mapping-by-Code)

Entity Framework/EF Core

- Entitätsklassen
 - nicht gemappte Properties müssen entspr. konfiguriert werden
- DbContext
 - enthält pro Entitätstyp ein `DbSet`
- Mapping
 - Per Code-Konventionen
 - FluentApi in `OnConfiguring` des DbContext
 - DataAnnotations

Ladestrategien

NHibernate

- Lazy Loading:
 - `.Load()`
 - Fetching-Strategie (Nachladen Abhängiger) per Konfiguration
 - Erzeugt Proxy
- Eager Loading:
 - `.Get()`
 - Alle(!) Abhängigen werden geladen (steuerbar)

Entity Framework/EF Core

- Entity Framework:
 - Konfigurierbar am Model
- EF Core:
 - Lazy Loading ist Default
 - `.Include/.ThenInclude` in Queries
 - oder
 - IdentityResolution (Preloading)
 - oder
 - nuget:
`Microsoft.EntityFrameworkCore.Proxies`
 - 1. Ebene wird nachgeladen

ChangeTracking/Daten persistieren

NHibernate

- Session kann Änderungen überwachen
- Session-Methoden:
 - **.Flush()** (auch `Transaction.Commit()`)
 - **.Save(Instanz)**
 - **.Update(Instanz)**
 - **.Evict(Instanz)**
 - **.Delete(Instanz)**

Entity Framework/EF Core

- **DbContext.ChangeTracker** überwacht Änderungen
- Einzige Methode zum Schreiben in die DB:
DbContext.SaveChanges()

Zeit für ne
DEMO

Primary Key – Werte-Generierung

NHibernate

- IDENTITY/Auto-Inkrement
- Sequence
- HI/LO-Algorithmen
(eigener + DB-spezifische)
- GUID
- Datum/Uhrzeit
- Manuell

Entity Framework/EF Core

- IDENTITY/Auto-Inkrement
- Sequence
- Manuell

Seit EF Core auch

- GUID
- Datum/Uhrzeit

Events und Interceptors

NHibernate

- Events
 - Session hat Events für nahezu jede Methode
- Interceptors
 - Abfrage und Manipulation von Objekten vor Datenbankoperationen

EF Core

- Events
 - Grundlegende Events in DbContext und ChangeTracker
- Interceptors
 - Manipulation von Datenoperationen
 - Ausführung von Kommandos auf der Datenbank

Fazit

NHibernate

- Pro
 - „gefühl“ leichtgewichtig und offen
 - gut kapselbar (Repository)
 - POCOs gut code-generierbar
 - native SQL-Unterstützung
 - vielfältige Erweiterungen durch Events und Interceptors
- Contra
 - virtual-Properties
 - Konkurrierende Mapping-Techniken (XML/FluentApi)
 - wenig Unterstützung für Code-First

EF Core

- Pro
 - Entitäten (POCOs) sind Klassen ohne techn. Abhängigkeiten (außer DbContext)
 - kapselbar (Repository/Unit-of-Work)
 - gutes Tooling
 - für Umsteiger von Entity Framework gewohnte Architektur
- Contra
 - „gefühl“ monolithisch
 - technisch abhängig vom DbContext
 - wenig Unterstützung für Database-First
 - ressourcenhungrig(?)



Bitte
nicht vergessen
zu bewerten!



Vielen Dank!

Slides und Code-Sample auf

<https://github.com/olaflischke/basta-spring-2025>

