Multi-Tier-Architektur

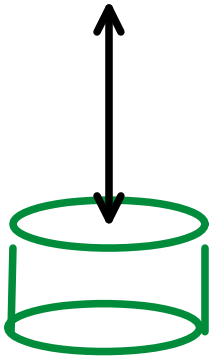
Mittwoch, 8. Februar 2023 08:45

UI

BL

DAL

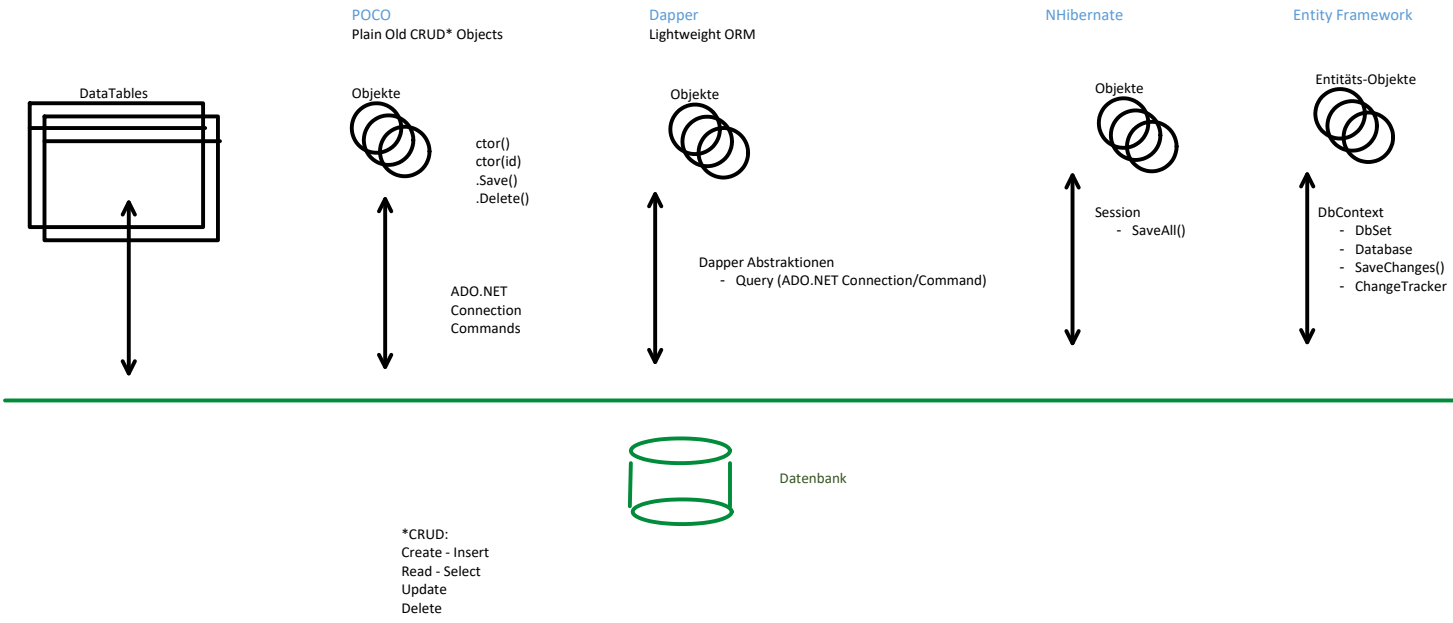
ADO.NET - DataTables
ADO.NET - Connections, Commands
Entity Framework



Datenbank

Data-Access-Layer

Mittwoch, 8. Februar 2023 09:35

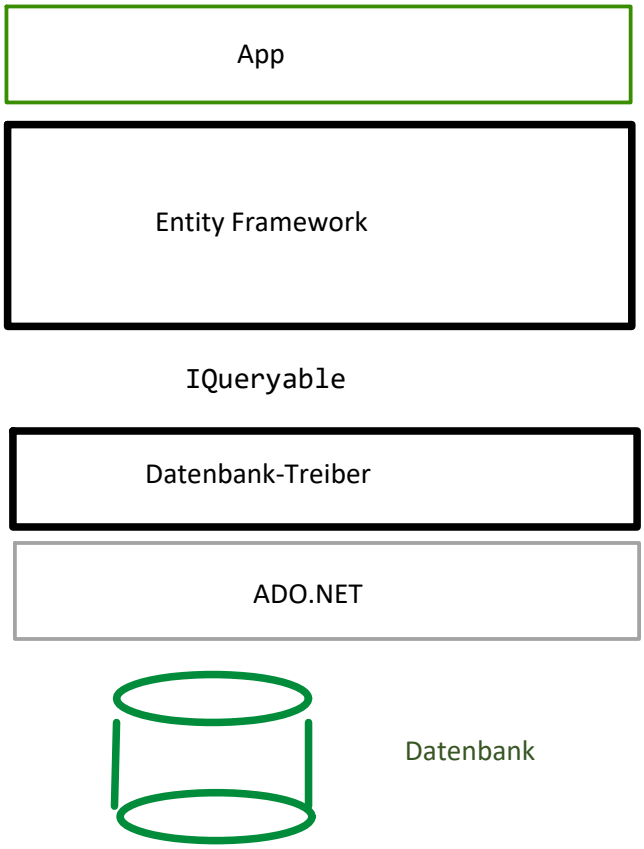


Datenbank

*CRUD:
Create - Insert
Read - Select
Update
Delete

Entity Framework Architektur

Mittwoch, 8. Februar 2023 09:59



Nuget
Microsoft.EntityFrameworkCore
Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Sqlite
Microsoft.EntityFrameworkCore.SqlServer
Npgsql.EntityFrameworkCore.PostgreSQL

Ausflug: .NET Framework vs. .NET (Core)

Mittwoch, 8. Februar 2023 10:36

.NET Framework (".NET Classic")

Jahr	Version	Bemerkungen
2000	1.0	
2001	1.1	Visual Studio .NET, ASP.NET (IIS)
2003	2.0	~650 Namespaces
2007	3.0	~~WPF, LINQ, Entity Framework~~
2008	3.5	WPF, LINQ, Entity Framework
2010	4.0	TPL (async/await), HTTP
2013	4.5	Roslyn-Compiler, ~14.000 Namespaces
2018	4.8.1	"Final Version"

.NET Standard

1.0
2.0
2.1
3.0/1

.NET Core

- plattformunabhängig
- leichtgewichtig, modular, RESTfull
- Open Source

Jahr	Version	Bemerkungen
2016	1.0	~850 Namespaces, Nuget, keine UI
	2.0	reines Backend
2019	3.0	WPF, Windows Forms - ~~Open Source~~
2020	5.0	.NET ~~Core~~
2021	6.0	Long Term Support (LTS)
2022	7.0	

Entity Framework Core

Mittwoch, 8. Februar 2023 13:59

Reverse Engineering (formerly known as Database First)

Datenbank existiert bereits, es soll ein Objektmodell für den Datenzugriff erstellt werden.

```
Scaffold-DbContext
-Connection "datasource=C:\ProgramData\SQLite\data\northwind.db"
-Provider Microsoft.EntityFrameworkCore.Sqlite
-OutputDir Model
-Context NorthwindContext
-Tables Customers, Orders, "Order Details", Products
```

Code First

Es wird eine Datenbank (automatisch?) generiert, die dem Objektmodell entspricht

Nullables in C#

Mittwoch, 8. Februar 2023 14:18

```
1  int a = 0;
2  int? a2 = null;
3  System.Nullable<int> a3 = null;
4
5  int b = a + (a2.HasValue ? a2.Value : 0);
6  int c = a + a2 ?? 0;
```

Unit Tests in Visual Studio

Mittwoch, 8. Februar 2023 15:10

Funktionen einer Klassenbibliothek gezielt aufrufen und ausführen

Vorgehen

- Projekt hinzufügen, Test-Vorlage (MSTest, Nunit, xUnit)
- Als [TestMethod] gekennzeichnete Methoden tauchen im Test-Explorer auf (Test/TestExplorer)
- Testexplorer: Tests einzeln, alle, oder Auswahl ausführen
- Statische Assert stellt Methoden bereit, das Testergebnis zu definieren

Unit Tests debuggen

Test als Debug starten

Context benutzen

Mittwoch, 8. Februar 2023 15:18

Zugriff auf die Datenbank:

- Instanz von DbContext
- DbSets des Contextes stellen die Daten bereit

Context per Code konfigurieren

```
DbContextOptions options = new DbContextOptionsBuilder<FolderContext>()
                                .UseSqlite("datasource=C:\\ProgramData\\SQLite\\data\\folders.db")
                                .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTrackingWithIdentityResolution)
                               .LogTo(...)
                                .Options;

FolderContext context = new FolderContext(options);
```

Language INtegrated Query

Mittwoch, 8. Februar 2023 15:25

SQL

```sql
SELECT * FROM Customers WHERE City LIKE 'Berlin%'
```

C# ohne LINQ

```
List<Customer> allCustomers = GetAllCustomers();

List<Customer> berliners = GetCustomersByCity(allCustomers, "Berlin");


private List<Customer> GetCustomersByCity(List<Customer> customers, string city)
{
      foreach (Customer cu in customers)
      {
            //...

```

C# mit LINQ

```
List<Customer> allCustomers = GetAllCustomers();

// Deklarative Syntax
var berliners1 = from cu in allCustomers
                 where cu.City.StartsWith("Berlin")
                 select new { cu.Vorname, cu.Nachname };

// Lambda-Syntax
var berliners2 = allCustomers.Where(cu => cu.City.StartsWith("Berlin"));

List<Customer> berls = berliners1.ToList();
```

Deferred Execution
Zugriff auf die Ergebnismenge führt Query (GetEnumerator() aus IEnumerable/IQueryable) aus

Wünsche

Mittwoch, 8. Februar 2023 16:08

- RawSql

DbContext - Datenoperationen

Donnerstag, 9. Februar 2023 16:30

Lesen

-> LINQ auf DbSets

- LINQ wird analysiert, daraus SQL gebaut

```
context.Customers.Where/Select/FirstOrDefault
```

Lazy Loading

-> DbContext lädt nur die allernotwendigsten Daten

```
context.Customers.Include(...)
                 .Include(...) // Customer
                 .ThenInclude(...) // Cascadierend

Zu ladende Daten im LINQ-Statement verwenden
```

Performance/Tweaking

- `.AsNoTracking` - laden ohne Tracking (ChangeTracker)
- `.AsEnumerable` - Query ausführen, nachfolgende Operationen im RAM

Context-Handling

Context sollte so sparsam wie möglich einsetzen!

- Context bevorzugt lokal instanziieren
- Tipp: using
- Tipp: Tracking nur dann, wenn notwendig (Speichern geplant)

Schreiben

Nur eine Methode, um in die Datenbank zu schreiben:
`context.SaveChanges();`

Änderungen werden erkannt mit Hilfe des ChangeTrackers

- `context.Entry(...).State - EntityState-Enum`
  Lässt sich auch per Code setzen, empfehlenswert ist jedoch passende Methode verwenden:
  `context.Customers.Attach(...)` statt `context.Entry(...).State = Modified`
- `SaveChanges` ändert den State zurück auf Unchanged
- `context.Entry(...).Reload()` - Daten neu laden, State = Unchanged

Code First: Model und Datenbank initialisieren

Freitag, 10. Februar 2023 09:50

Model erstellen

- Nuget-Pakete für EF + Treiber
- Context-Klasse
  - die DbContext beerbt
  - DbSet für jeden Typ, der eine Tabelle in der DB darstellt (Entitätsklassen)
- Entitätsklasse
  - Property, die PK sein kann
    - Passender Typ: int, string, GUID,...
    - Namenskonvention: Id, [Klasse]Id, [Klasse]_Id
  - Relations
    - 1-Seite: Property von passenden Typ
    - n-Seite: Collection des passenden Typ
  - Parameterloser Konstruktor

Datenbank erstellen

```
context.Database.EnsureDeleted(); // Datenbank löschen, wenn vorhanden
context.Database.EnsureCreated(); // Datenbank anlegen, wenn nicht vorhanden
context.Database.Migrate()
```

Daten schreiben

- Modell mit Daten füllen
- `context.[DbSet].Add(...)/.AddRange(...)`
- `context.SaveChanges()`

```
if (context != null)
{
    context.Folders.Add(root);
    context.SaveChanges();
```

Migrations (Code First)

Freitag, 10. Februar 2023 14:40

Migrations gleichendas Datenbankschema an das Objektmodellschema an.

PMC-Commands

- `Add-Migration`
  - erstellt eine Migration auf der Basis der letzten Migration
  - wenn keine Migration vorh., erstellt initiale Migration
- `Remove-Migration`
  - löscht Migration
- `Update-Database [-Migration Ziel]`
  - aktualisert __EFMigrationHistory-Tabelle
  - ohne Angabe einer Zielmigration: Führt alle Migrations durch, die noch nicht durchgeführt wurden
  - Zielmigration: Führt alle Migrationen vom aktuellen Stand bis zur Zielmigration aus (in beide Richtungen!)

Sonderfall EnsureCreated()

Problem: Durch EnsureCreated() existiert die DB bereits!
1. Migration erzeugt mit Add-Migration erzeugt jedoch eine Migration, die von null ausgeht -> Update-Database läuft in Fehler

Lösung 1: Datenbank löschen, im Code Migrate()-Methode verwenden für 1. Migration
Lösung 2: Name der Migration-Datei ohne .cs in die __EFMigrationHistory-Tabelle als ersten Eintrag schreiben (Tipp: __EFMigrationHistory wird auch erzeugt, wenn Update-Database fehlschlägt)