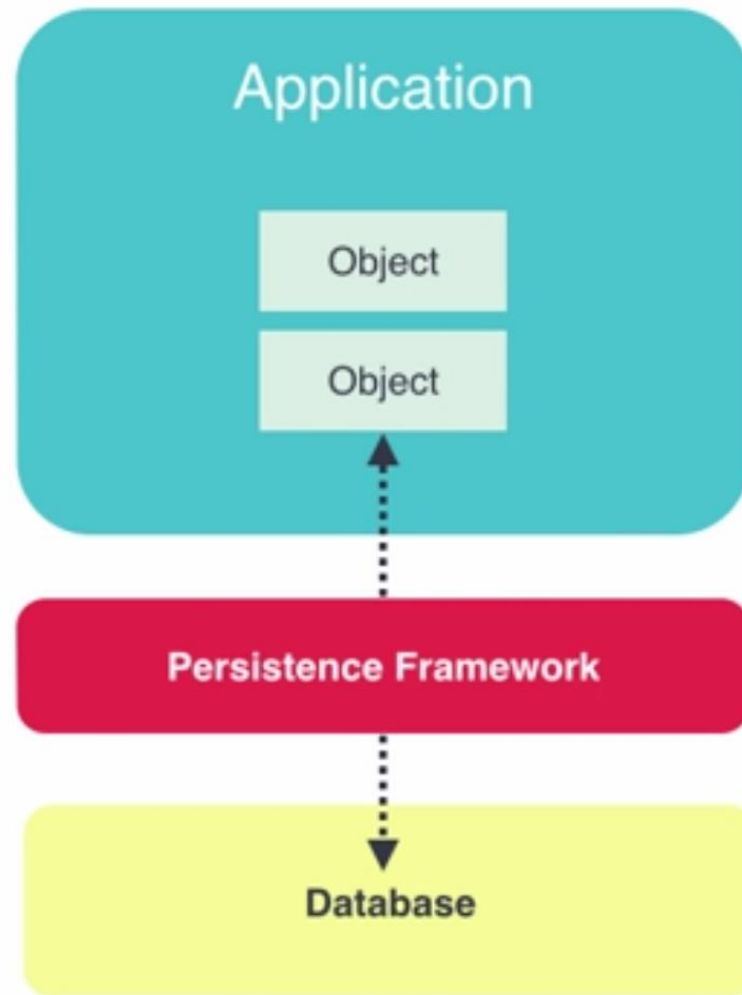


Entity Framework (EF)

Lernziele

- Sie verstehen die Grundlagen eines OR-Mappers am Beispiel von EF
- Sie kennen die Mappingkonzepte (OR-Mapping) von EF
- Sie kennen die Funktionalitäten der Object-Services (DbContext)
- Sie kennen die CRUD-Operationen (inkl. LINQ)
- Sie können einfache Modelle erstellen und LINQ-Abfragen formulieren

Was ist Entity Framework?



Versionsgeschichte

- Version 1.0: 11.08.2008
- Version 4.0: 12.04.2010
- Version 4.1: 13.04.2011 (Code First)
- Version 4.3.1: Februar 2012 (Unterstützung von Migration)
- Version 5.0: August 2012 (.NET 4.5)
- Version 6.0: Oktober 2013 (Verbesserung Code First, OpenSource)
 - Aktuelle Version: 6.4.4 (15. Mai 2020)
- EFCore 1.0: Juni 2016 (kompletter Rewrite)
- EFCore 2.0: August 2017
- EFCore 3.0: September 2019
- Weiter: <https://learn.microsoft.com/en-us/ef/core/what-is-new/>
- EFCore bietet generell mehr Features als EF6, es fallen aber auch einige weg
 - <https://www.softwareblogs.com/Home/Blog/how/DiffEF6andCore/entity-framework-6-entity-framework-core-difference-compare-examples>
 - <https://docs.microsoft.com/en-us/ef/efcore-and-ef6/>

Modellierungsansätze (1)

Code First

- Erstellung der Domain-Klassen
- EF generiert Datenbanktabellen

Database First

- Modellierung der Tabellen
- EF generiert Domain-Klassen
- scaffolding

~~Model First~~

- ~~Erstellung eines UML-Diagramm~~
- ~~EF generiert Domain-Klassen und Datenbanktabellen~~

CODE FIRST

Vorgehen

- Console App (.NET Core)
- Installation Entity Framework (Package Manager)
- Entityklassen erstellen
- DbContext erstellen
- ConnectionString spezifizieren

```
{  
  "ConnectionStrings": {  
    "EFCoreDemo": "Data Source=.\symas; Database=EFCoreDemo; Trusted_Connection=True;"  
  }  
}
```

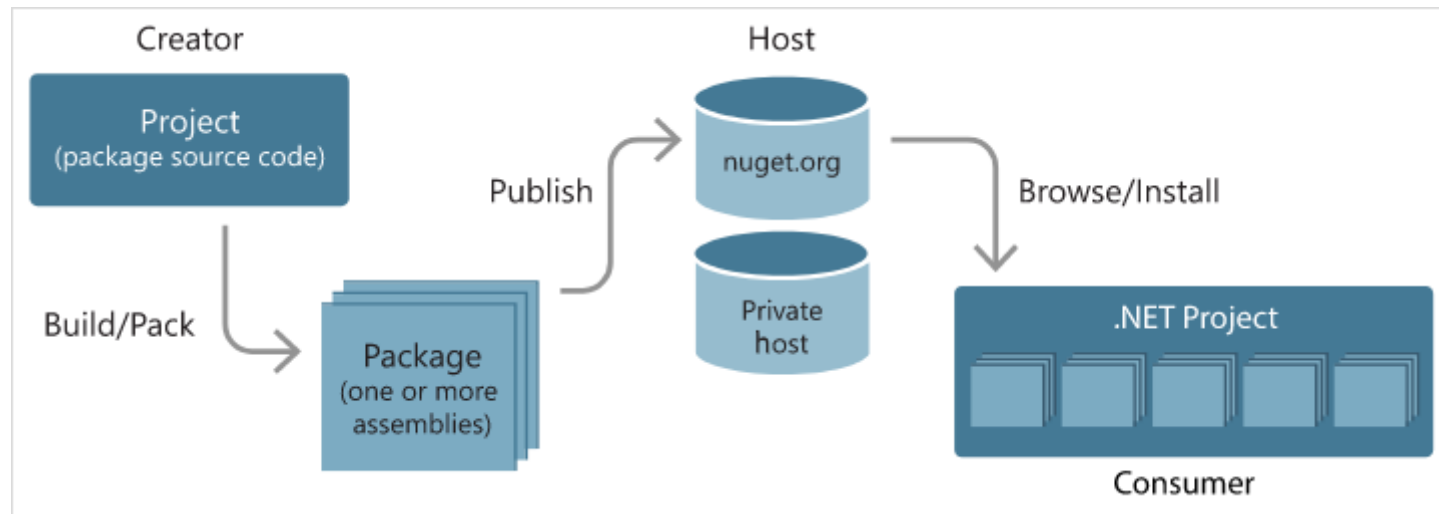
oder

```
optionsBuilder.UseSqlServer("Data Source=.\symas; Database=EFCoreDemo; Trusted_Connection=True;");
```

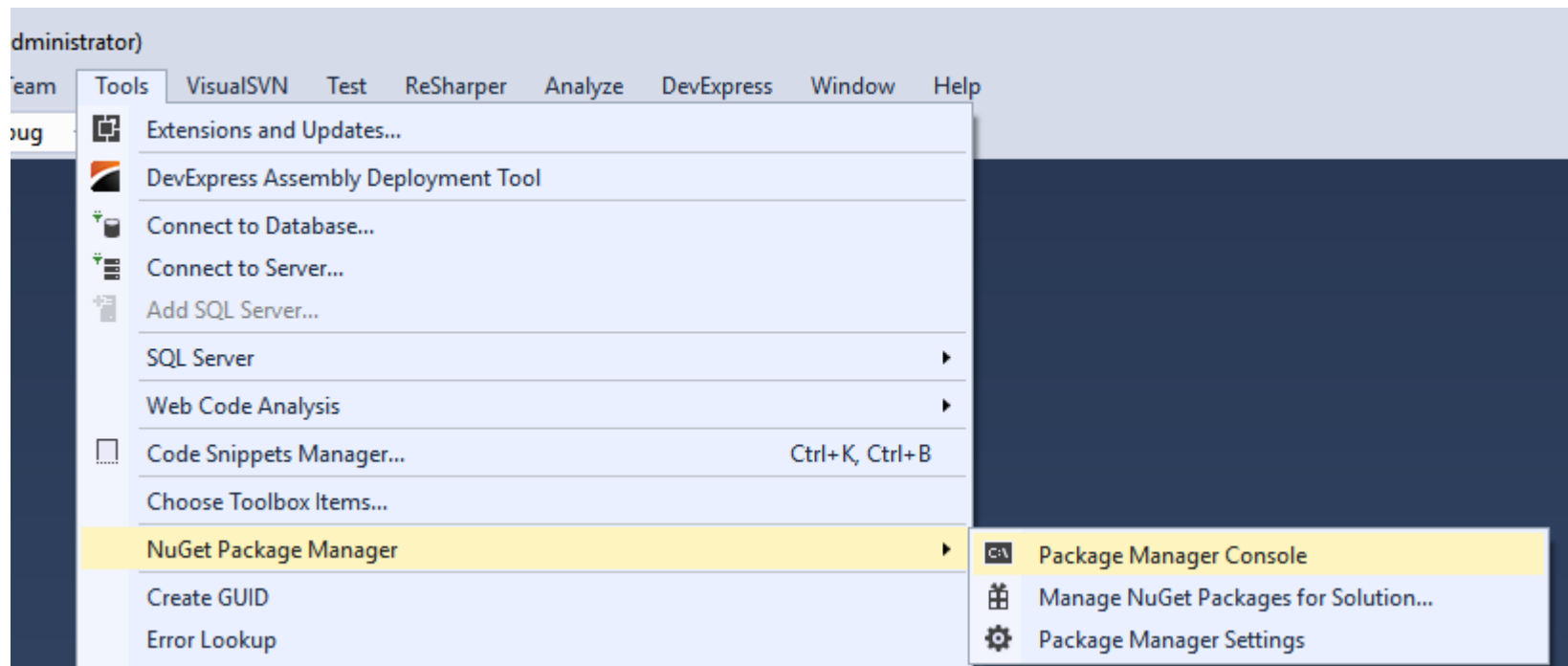
- PM> enable-migrations
- PM> add-migration InitialModel
- PM> update-database -verbose

Exkurs: NuGet

- Paketverwaltung
- Einführung 2010



Installation Entity Framework



```
PM> install-package Microsoft.EntityFrameworkCore.SqlServer
```

```
PM> install-package Microsoft.EntityFrameworkCore.Tools
```


Model

- Ordner Model erstellen
- Für jede Entity eine Klasse
- DbContext erstellen
- `PM> Add-Migration InitialCreate`
- `PM> Update-Database`
- `PM> Remove-Migration`
 - Entfernt die letzte Migration

DataAnnotations vs. FluentAPI

- <https://dotnetcoretutorials.com/2020/06/27/a-cleaner-way-to-do-entity-configuration-with-ef-core/>

DbContext

- Persistente Objekte sind immer einem Kontext zugeordnet
 - Stellt die Objectservices zur Verfügung
 - Verwaltet die Verbindungen zur Datenbank
 - Kapselt die Datenbanksessions
 - Generiert SQL-Abfragen gegen die Datenbank (CRUD)
 - Speichert Änderungen an Daten bzw. neue Daten (CRUD)
 - Serialisieren und materialisieren von Entitätsobjekten
 - Lokaler Cache um die materialisierten Objekte zu speichern
 - Änderungsverfolgung (Change Tracking)
 - Management von Nebenläufigkeiten
-  **sollte nur für eine sehr kurze Zeit existieren! (UnitOfWork)**

Bestandteile eines DbContext

```
public class CourseContext : DbContext {  
  
    protected override void OnConfiguring(  
        DbContextOptionsBuilder optionsBuilder) {  
  
        //optionsBuilder.UseSqlServer("Data Source=.\syms;  
        //    Database=EFCoreDemo; Trusted_Connection=True");  
  
        var configuration = new ConfigurationBuilder()  
            .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)  
            .AddJsonFile("appsettings.json")  
            .Build();  
  
        optionsBuilder.UseSqlServer(  
            configuration.GetConnectionString("EFCoreDemo"));  
  
        optionsBuilder.LogTo(Console.WriteLine);  
    }  
}
```

Der konkrete Kontext erbt von DbContext

ConnectionString der verwendet werden soll (siehe appsettings.json)

Logging auf Console

```
public virtual DbSet<Course> Courses { get; set; }  
public virtual DbSet<Author> Authors { get; set; }  
}
```

DbSet entspricht einer Tabelle in der Datenbank

ConnectionString

```
{  
  "ConnectionStrings": {  
    "EFCoreDemo": "Data Source=.\symas;  
                  Database=EFCoreDemo; Trusted_Connection=True;"  
  }  
}
```

Erzeugen einer neuen Entität

`using (var context = new CourseContext()) {` Kontext erstellen – Achtung: Lebensdauer beachten!

- Öffnet eine Datenbanksession
- Initialisiert Cache und Änderungsverfolgung

```
var author = new Author() {  
    Name = "Thomas Kehl"  
};
```

Entität erzeugen

```
var course = new Course() {  
    Title = "C# Programming",  
    Author = author  
};
```

```
context.Courses.Add(course);
```

Entität zu DbSet hinzufügen. Gehört nun zur Tabelle im Memory.

```
context.SaveChanges();
```

Änderungen in der Datenbank speichern

```
}
```

Kontext schliessen

- Löscht den Cache
- Schliesst die Datenbankverbindung

Datenbankstatements ausgeben

```
protected override void OnConfiguring(
    DbContextOptionsBuilder optionsBuilder) {

    var configuration = new ConfigurationBuilder()
        .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
        .AddJsonFile("appsettings.json")
        .Build();

    optionsBuilder.UseSqlServer(
        configuration.GetConnectionString("EFCoreDemo"));

    optionsBuilder.LogTo(Console.WriteLine);
}
```

Neue Klasse

```
public class Category {  
    // Ansonsten ist Id AutoIncrement  
    [DatabaseGenerated(DatabaseGeneratedOption.None)]  
    public int Id { get; set; }  
    public string Name { get; set; }  
}
```

```
public class CourseContext : DbContext {  
    ...  
    public DbSet<Category> Categories { get; set; }  
    ...  
}
```

```
PM> add-migration AddCategoriesTable
```

```
migrationBuilder.Sql("INSERT INTO Categories VALUES (1, 'Web Development')");  
migrationBuilder.Sql("INSERT INTO Categories VALUES (2, 'Programming Languages')");
```

```
PM> update-database
```

Alternative:

```
context.Database.Migrate(); // benötigt using Microsoft.EntityFrameworkCore;
```


Downgrade

```
PM> update-database -Migration AddCategoriesTable
```

Nun werden alle Migrations, die nach `AddCategoriesTable` ausgeführt wurden, wieder rückgängig gemacht (Methode `Down()`).

CodeFirst - Conventions

- Conventions sind Regeln, welche anhand den Entity-Klassen das Conceptual Model definieren
- PrimaryKeys, ForeignKeys, Datentypen für Columns usw. werden anhand den Conventions aus den Entity-Klassen abgeleitet
- Siehe <https://www.entityframeworktutorial.net/efcore/conventions-in-ef-core.aspx>

Conventions überschreiben

- Annotations

```
[Table("Courses")]
public partial class Course {
    public int Id { get; set; }

    [Required]
    public string Description { get; set; }
}

public partial class CourseDescriptionRequired : DbMigration {
    public override void Up() {
        AlterColumn("dbo.Courses", "Description", c => c.String(nullable: false));
    }

    public override void Down() {
        AlterColumn("dbo.Courses", "Description", c => c.String());
    }
}
```

weitere Informationen:

<http://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx>

Conventions überschreiben

- FluentAPI

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {  
    modelBuilder.Entity<Course>()  
        .Property(t => t.Description)  
        .IsRequired();  
}
```

```
public partial class CourseDescriptionRequired : DbMigration {  
    public override void Up() {  
        AlterColumn("dbo.Courses", "Description", c => c.String(nullable: false));  
    }  
  
    public override void Down() {  
        AlterColumn("dbo.Courses", "Description", c => c.String());  
    }  
}
```

weitere Informationen:

<http://www.entityframeworktutorial.net/code-first/fluent-api-in-code-first.aspx>

Navigation Properties

- <https://www.learnentityframeworkcore.com/relationships#navigation-properties>
 - Sowie nachfolgende Kapitel
- One-To-Many
 - Reference Navigation Property (z.B. Category)
 - Optional: Nullable
 - U/O ForeignKey-Property (CategoryId)
 - Collection Navigation Property
- Many-To-Many
 - Im Gegensatz zu EF6 muss die Matrix-Tabelle definiert werden
 - Konfiguration der Beziehung in OnModelCreating (siehe CourseTag)

```
public class Course {  
    // ...  
    public virtual ICollection<CourseTag> CourseTags { get; set; }  
    public virtual Category Category {get; set;}  
  
}
```

- One-To-One

Data Seeding

- Initiale Daten für
 - Lookup-Tabellen
 - Stammdaten
 - Demodaten
- Erstellung mittels Migrations-Script (OnModelCreating() - Update und Revert möglich)

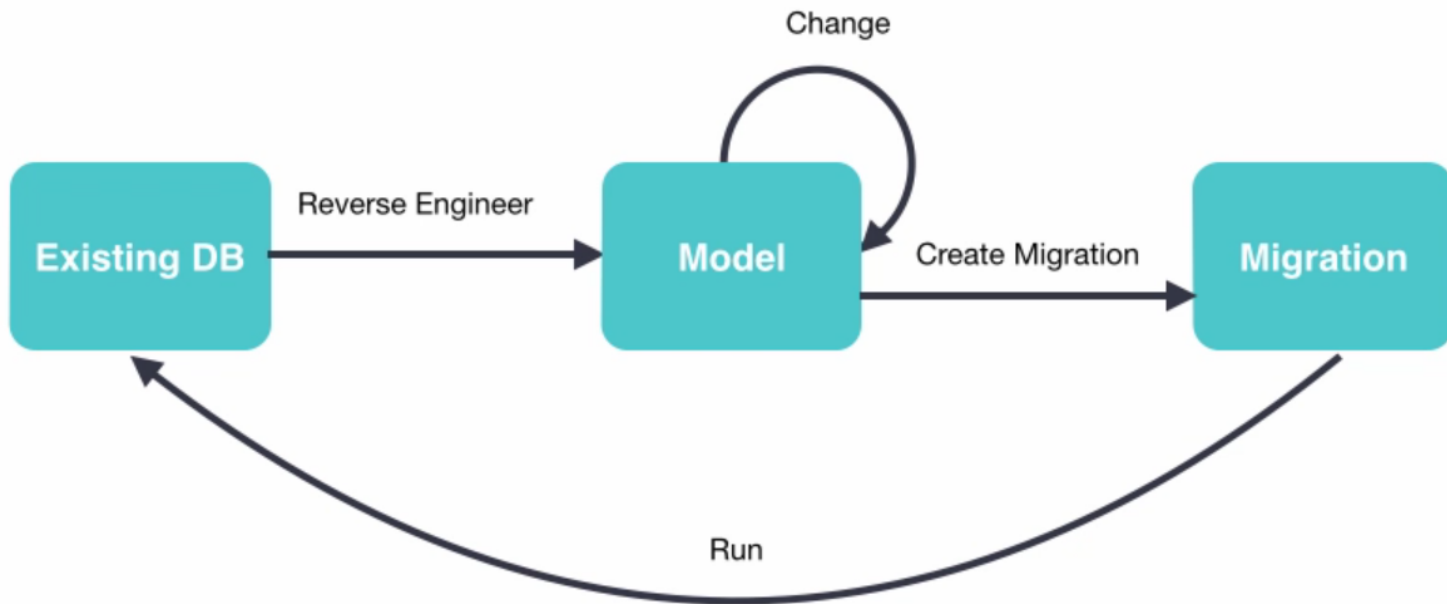
```
public override void OnModelCreating(ModelBuilder modelBuilder) {  
    modelBuilder.Entity<Author>().HasData(new Author() {Id = 10, Name = "Max"});  
}
```

In Migration:

```
migrationBuilder.InsertData(  
    table: "Authors",  
    columns: new[] { "Id", "City", "Name", "Phone" },  
    values: new object[] { 10, null, "Max", null });  
  
migrationBuilder.DeleteData(  
    table: "Authors",  
    keyColumn: "Id",  
    keyValue: 10);
```

CodeFirst mit einer bestehenden Datenbank

- Scaffolding



- <https://docs.microsoft.com/en-us/ef/core/managing-schemas/scaffolding?tabs=vs>

Migrations in Production

```
PM> Script-DbContext
```

```
PM> Script-Migration
```

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/powershell>

oder

```
context.Database.Migrate();
```

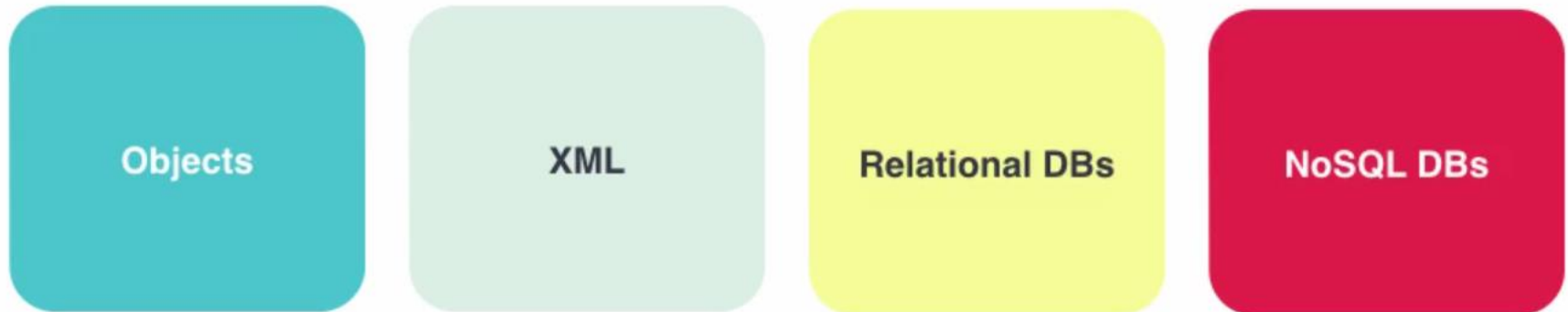

Übungsaufgaben

- Lösen Sie die Übungsaufgaben zu Code First.

DATENABFRAGE

LINQ

- **L**anguage **I**ntegrated **Q**uery
- Eingeführt von Microsoft mit dem Ziel, die Lücke zwischen der Objektwelt und Datenwelt zu schliessen
- Abfrage jedes Datenstores – vorausgesetzt LINQ-Provider



Vor LINQ

SQL Server



T-SQL

```
USE Northwind;  
GO  
SELECT * FROM Customers  
WHERE IsGold = 1  
ORDER BY Country
```

Oracle



PL/SQL

```
Declare  
NUM number:=1;  
Sum number:=0;  
begin  
loop  
NUM1 := NUM+2;  
Sum:=Sum+Num1;  
exit when NUM1=100;  
end loop;  
end;
```

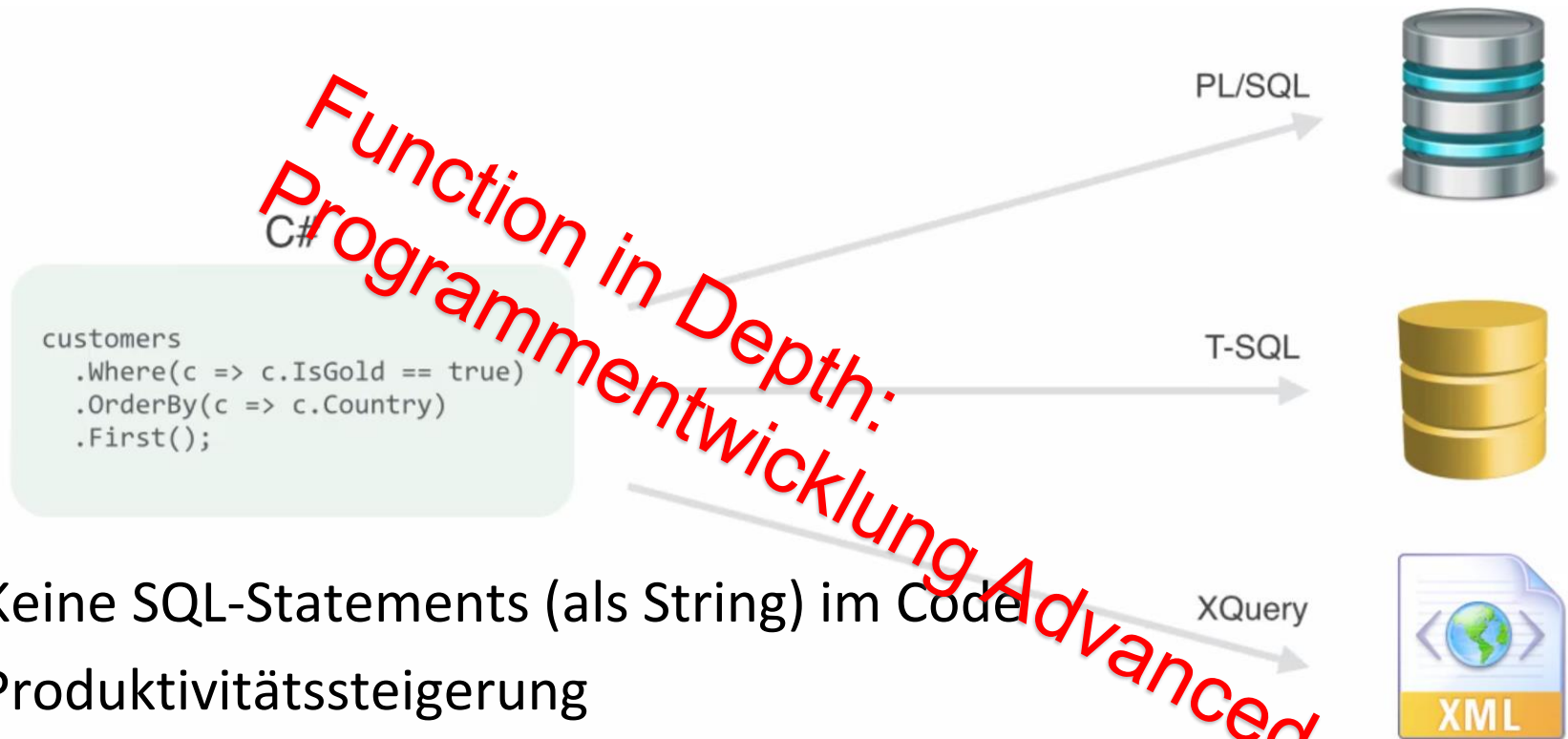
XML



XQuery

```
./actors/actor[ends-with(., 'Lisa')]
```

Mit LINQ



- Keine SQL-Statements (als String) im Code
- Produktivitätssteigerung
- Für einfache Queries – sobalds komplex wird, wird LINQ unübersichtlich und langsam. Besser: Stored Procedure

LINQ in Action

```
var query =  
    from c in context.Courses  
    where c.Title.Contains("c#")  
    orderby c.Title  
    select c;
```

```
foreach (var course in query) {  
    Console.WriteLine(course.Title);  
}
```

```
Opened connection at 04.12.2017 16:58:08 +01:00  
SELECT  
    [Extent1].[Id] AS [Id],  
    [Extent1].[Name] AS [Name],  
    [Extent1].[Description] AS [Description],  
    [Extent1].[Level] AS [Level],  
    [Extent1].[FullPrice] AS [FullPrice],  
    [Extent1].[AuthorId] AS [AuthorId]  
FROM [dbo].[Courses] AS [Extent1]  
WHERE [Extent1].[Name] LIKE N'%c#%'  
ORDER BY [Extent1].[Name] ASC  
  
-- Executing at 04.12.2017 16:58:08 +01:00  
-- Completed in 1 ms with result: SqlDataReader  
  
A 16 Hour C# Course with Visual Studio 2013  
C# Advanced  
C# Basics  
C# Intermediate  
Closed connection at 04.12.2017 16:58:08 +01:00
```

Extension Methods

```
var courses = context.Courses
    .Where(c => c.Title.Contains("c#"))
    .OrderBy(c => c.Title);
```

```
foreach (var course in query) {
    Console.WriteLine(course.Title);
}
```

- Leistungsfähiger bzgl. Möglichkeiten als Query-Syntax

```
Opened connection at 04.12.2017 16:58:08 +01:00
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[Description] AS [Description],
    [Extent1].[Level] AS [Level],
    [Extent1].[FullPrice] AS [FullPrice],
    [Extent1].[AuthorId] AS [AuthorId]
FROM [dbo].[Courses] AS [Extent1]
WHERE [Extent1].[Name] LIKE N'%c#%'
ORDER BY [Extent1].[Name] ASC

-- Executing at 04.12.2017 16:58:08 +01:00
-- Completed in 1 ms with result: SqlDataReader

A 16 Hour C# Course with Visual Studio 2013
C# Advanced
C# Basics
C# Intermediate
Closed connection at 04.12.2017 16:58:08 +01:00
```

Deferred Execution der Queries

- Iteration über query-Variable
- Aufruf von `ToList()`, `ToArray()`, `ToDictionary()`
- Aufruf von `First()`, `Last()`, `Single()`, `Count()`, `Max()`, `Min()`, `Average()`
- Queries können erweitert werden

```
var courses = context.Courses;
var filtered = courses.Where(c => c.Level == 1);
var sorted = filtered.OrderBy(c => c.Name);
foreach (var course in sorted) { // SQL-Statement wird erst
                                // hier ausgeführt!
    Console.WriteLine($"Kurs: {course.Name}");
}
```


Find

```
var course = context.Courses.Find(4);
```

- Lädt das Entity anhand des Primärschlüssels
- Sucht zuerst im Cache des DbContext
- Wenn im Cache nicht vorhanden, wird ein SQL-Query ausgeführt
- Für zusammengesetzte Primärschlüssel:
`Find(params object[] keyValues)`

LOADING RELEATED OBJECTS

Lazy Loading

```
var author = context.Authors.Single(c => c.Id == 2);
```

erste Datenbankabfrage

```
SELECT ... FROM  
Authors WHERE Id=2
```

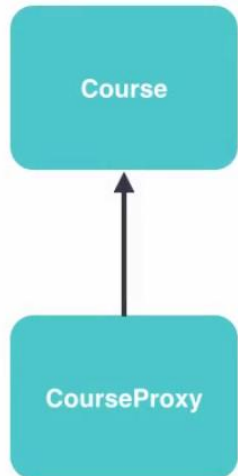
```
foreach (var c in author.Courses) {  
    Console.WriteLine(c.Title);  
}
```

zweite Datenbankabfrage

```
SELECT ... FROM  
Courses WHERE ...
```

Lazy Loading - Funktionsweise

```
public class Author {  
    ...  
    public virtual ICollection<Course> Courses { get; set; }  
    ...  
}
```



```
public class AuthorProxy : Author {  
    ...  
    public override ICollection<Course> Courses {  
        get {  
            if (_courses == null) {  
                _courses = context.LoadCourses();  
            }  
            return _courses;  
        }  
    }  
    ...  
}
```

Vereinfacht dargestellt!

Name	Value	Type
author	{Castle.Proxies.AuthorProxy}	EFCoreDemo.Models.Author {Castle.Proxies.AuthorProxy}

Lazy Loading – N+1-Problem

- Das Laden von N Entities und deren verbundenen Entities resultiert in N+1 Datenbankabfragen

```
var courses = context.Courses.ToList();
```

erste Datenbankabfrage

```
SELECT ... FROM Courses WHERE ...
```

```
foreach (var c in courses) {
```

```
    Console.WriteLine($"{c.Title}-{c.Author.Name}");
```

Für jeden Kurs wird 1x das Property Author initialisiert (Lazy Loading)

```
}
```



Lazy Loading mit Vorsicht verwenden!!!

Eager Loading

Magic String!!!

```
var courses = context.Courses.Include("Author").ToList();  
foreach (var c in courses) {  
    Console.WriteLine($"{c.Title} - {c.Author.Name}");  
}
```

Viel besser:

```
using System.Data.Entity;  
...  
var courses = context.Courses.Include(c => c.Author).ToList();  
foreach (var c in courses) {  
    Console.WriteLine($"{c.Title} - {c.Author.Name}");  
}  
  
// Eager Loading für einfache Eigenschaften  
var courses3 = context.Courses.Include(c => c.Author.Address)  
  
// Eager Loading für Collection-Properties  
var courses3 = context.Courses.Include(c => c.Tags.Select(t => t.Moderator));
```

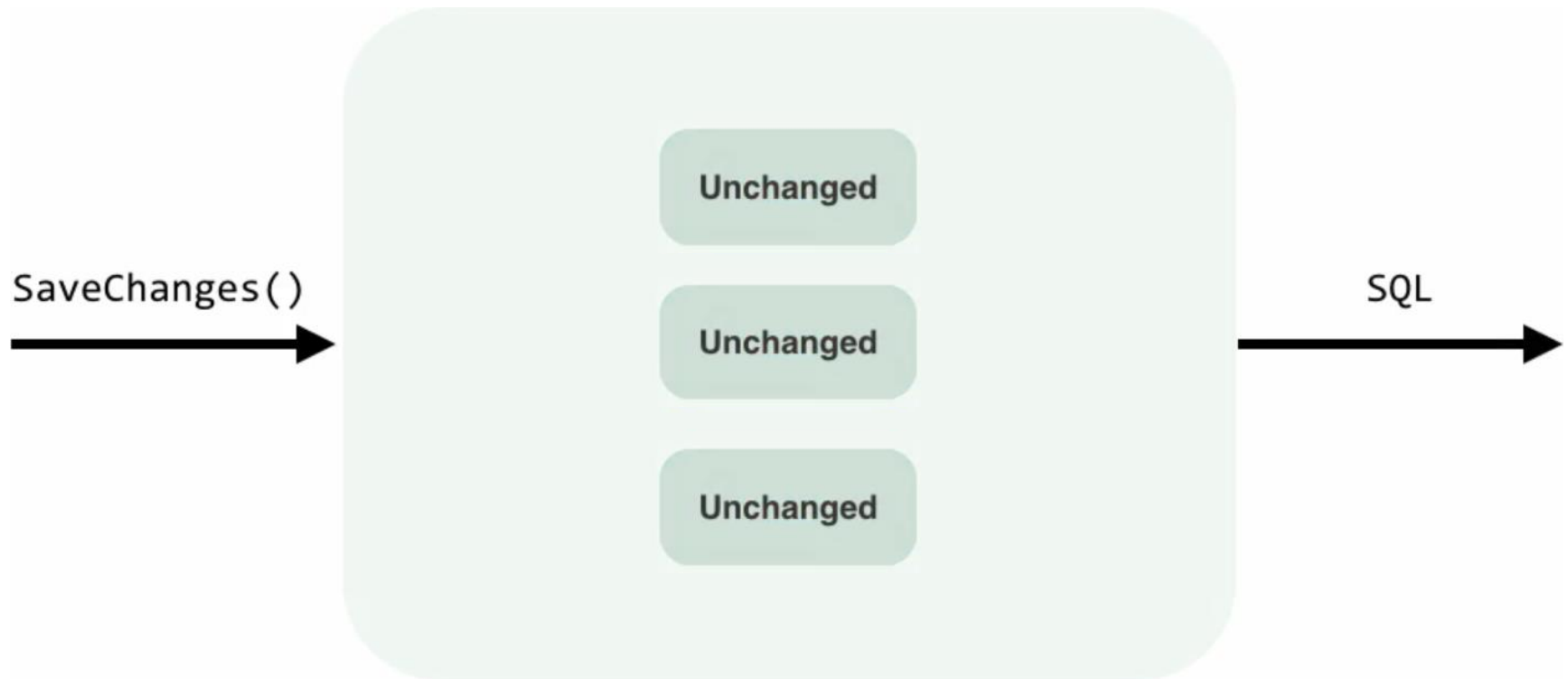
Loading Related Data

<https://docs.microsoft.com/en-us/ef/core/querying/related-data/>

DATEN ÄNDERN

Change Tracking

- Verantwortlich für die Verfolgung der Zustände der Objekte im DbContext



Daten aktualisieren

```
var course = context.Courses.Find(4);
```

Entity in den Context laden.

State = unchanged

```
course.Name = "New Name";
```

Update einzelner Eigenschaften

State = modified

```
course.Author = context.Authors.Find(2);
```

Autor zuweisen

```
context.SaveChanges();
```

UPDATE Courses SET Name=..., AuthorId=...
WHERE Id=4

Daten löschen (1)



```
var author = context.Authors.Find(2);
```

Entity in den Context laden.
State = unchanged

```
context.Authors.Remove(author);
```

Entity entfernen
State = deleted

```
context.SaveChanges();
```

DELETE FROM Authors WHERE Id=2
→ DbUpdateException
 → wrapped UpdateException
 → wrapped SqlException

Daten löschen (2)

```
author = context.Authors  
    .Include(a => a.Courses)  
    .Single(a => a.Id == 2);
```

Entity in den Context laden.
inkl. Referenzen (Eager Loading)
State = unchanged

```
context.Courses.RemoveRange(author.Courses);  
context.Authors.Remove(author);
```

Liste der Referenzen entfernen
Entity entfernen
State = deleted (gilt für Entity und Referenzen)

```
context.SaveChanges();
```

```
DELETE FROM Courses WHERE Id=4  
DELETE FROM Courses WHERE Id=5  
DELETE FROM Authors WHERE Id=2
```

Arbeiten mit dem ChangeTracker

- Debugging
- Auditing

```
// Add object
context.Authors.Add(new Author() {Name = "New Author"});
// Update object
var author = context.Authors.Find(3);
author.Name = "updated";
// Remove object
var another = context.Authors.Find(4);
context.Authors.Remove(another);

var entries = context.ChangeTracker.Entries();
foreach (var entry in entries) {
    // entry.Reload();
    Console.WriteLine(entry.State);
}
```

entry ist vom Typ `DbEntityEntry`

- `CurrentValues`
- `OriginalValues`
- `State`

`entry.Reload()` lädt das Entity neu von der Datenbank

Übungsaufgaben

- Lösen Sie die Übungsaufgaben zu Queries/Loading.

VERWENDUNG IN EINER APPLIKATION

Repository (1)

Martin Fowler:

Mediates between the domain and data mapping layers, acting like an **in-memory collection** of domain objects.

Vorteile:

- Minimiert duplizierte Abfrage-Logik

```
var topSellingCourses = context.Courses
    .Where(c => c.IsPublic && c.IsApproved)
    .OrderByDescending(c => c.Sales)
    .Take(10);
```

```
var courses = repository.GetTopSellingCourses(category, count);
```


Repository (2)

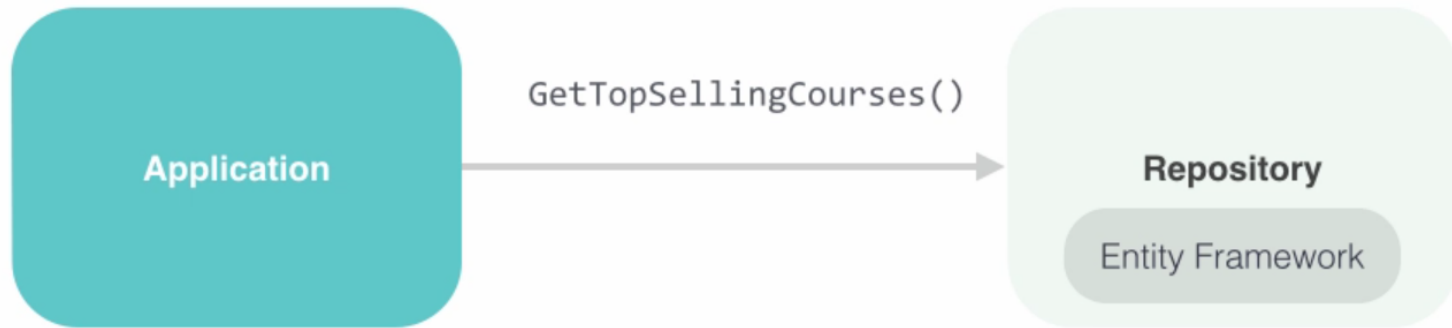
Vorteile:

- Entkoppelt die Applikation vom Persistence Framework

Ein neues O/RM alle 2 Jahre!

- ADO.NET
- LINQ to SQL
- Entity Framework v1
- nHibernate
- Entity Framework v4
- Entity Framework v4.1: DbContext
- EF Core 1.0: complete re-write

Repository (3)



Robert C. Martin:

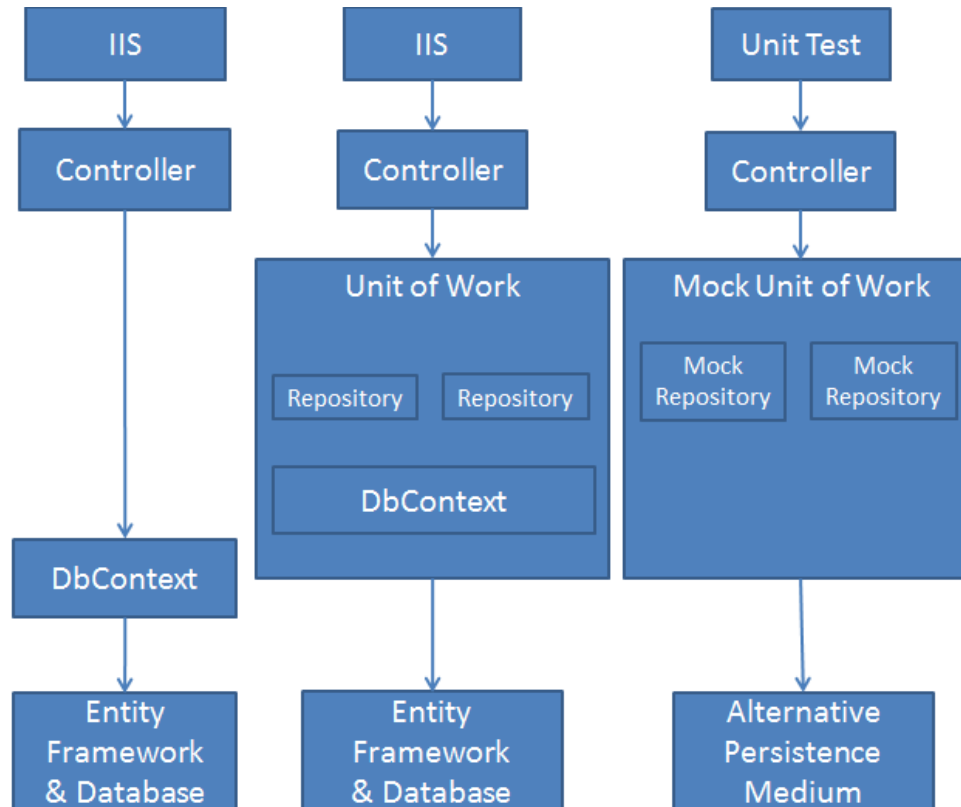
Clean Architecture

The Architecture should be independent of frameworks.

Repository mit UnitOfWork

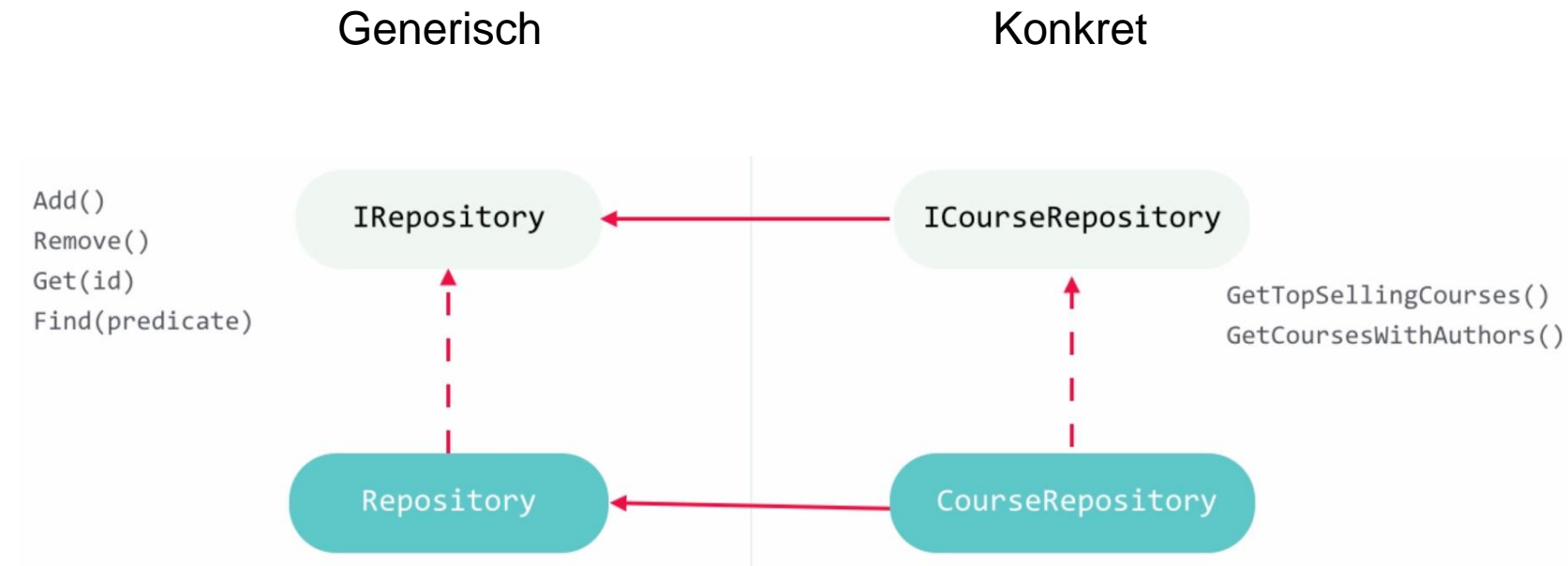
No Repository

With Repository

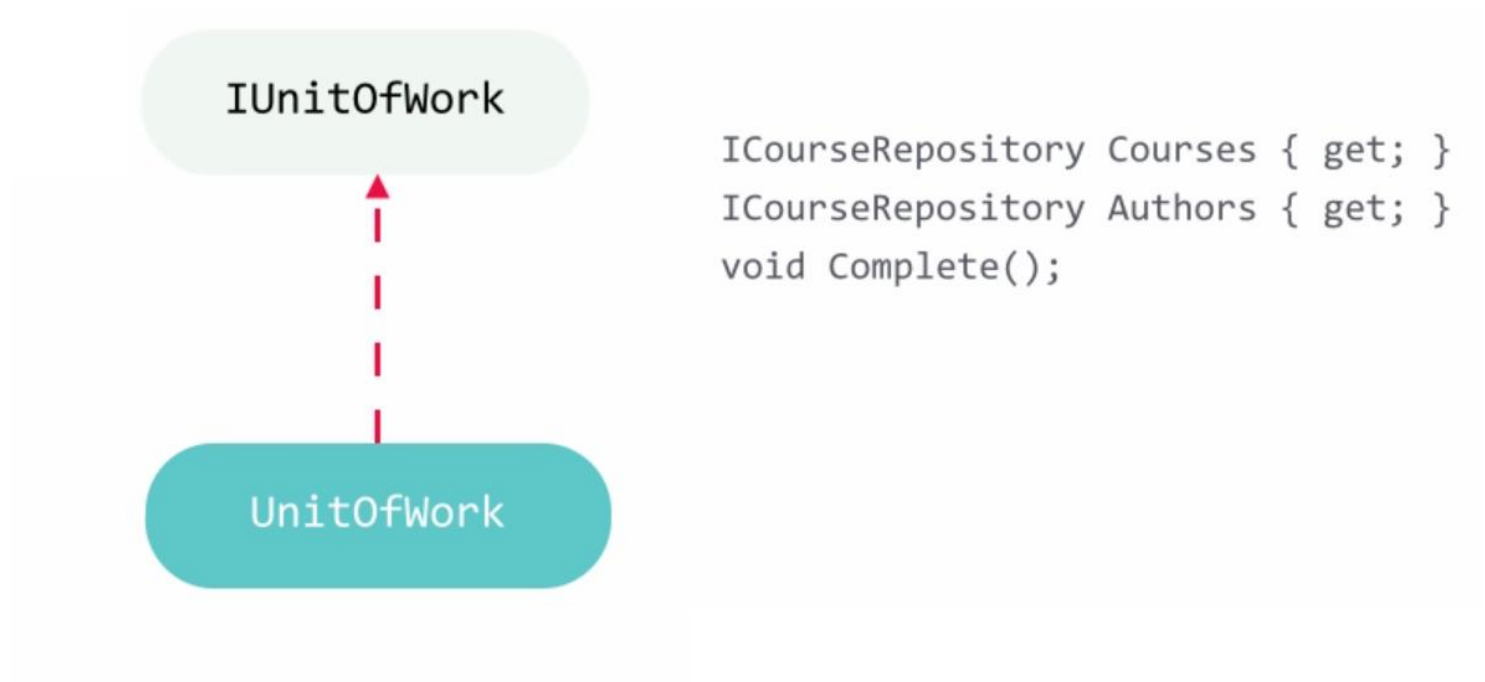


Quelle: Microsoft

Implementierung Repository



Implementierung Unit Of Work



In Action ...

- siehe RepositoryDemo

Architektur

Presentation

Modul
«Software-Architektur und Design»

Business Logic / Core

Data Access

Selbststudium

- Übungen zu EntityFramework
- Semesterprojekt