Entity Framework Core

Entity Framework ist ein **ORM** – **object relational mapper**, und überbrückt die Unterschiede zwischen Objektorientiertem Code und den Entitäten und ihren Beziehungen in einer SQL Datenbank.

Dabei verwaltet es zusätzlich den **Lifecycle** der **Objekte** und etwaige Änderungen.

Übersicht der Konzepte

1. DbContext
2. Migrationen
3. POCOS und Definitionen
4. ChangeTracking
5. Code first und Database first Ansätze

1 DbContext

Der DbContext hat mehrere Funktionen. Die wichtigsten high level sind:

1. Repräsentiert eine **Session** mit der **Datenbank**
    a. Connection
    b. Transaktionsmanagement
    c. Absetzen von Commands und Queries
2. Verwaltet die Entities, welche die Tabellen repräsentieren
    a. Properties vom Typ **DbSet<T>** erlauben Zugriff auf die Db Command und Query Methoden der Entities/Tabellen
    b. Änderungen an Properties und Beziehungen werden über den DbContext verwendet um **Migrationen** zu gestatten

Zusammengefasst ist der DbContext als zentrales Konzept also dafür verantwortlich, die Session mit der Datenbank zu abstrahieren und die Repräsentation der Entities und zugehöriger Migrationen zu übernehmen.

Die Session können wir wie folgt ganz einfach erstellen, wenn der Context korrekt erstellt wurde. (siehe das Video zum ersten Eindruck mit database first für details)

```
using (var context = new BasicsContext())
{
    var items = context.Entity.ToList();
    string result = string.Join("\n", items.Select(e => $"Id: [{e.Id}] - Name: [{e.Name}]"));
    System.Console.WriteLine(result);
}
```

Abbildung 1 - Nutzen des DbContext

2 Migrationen

Migrationen dienen dem **Zweck** das **Datenbankschema** zu **verwalten**. Dafür erstellen wir Migrationen, die in **Migrationsdateien** gespeichert werden. Diese können dann im Quellcodeverwaltungssystem eingecheckt werden.

Zudem sichert diese **einen Single point of truth** für das Datenbankschema.

Dies wird unter Verwendung der **POCOs** und des **DBContext** sowie des **dotnet ef core CLI** Tools bewerkstelligt. Die technischen Details lernen wir im Kapitel über Migrationen.

3 POCOS und Definitionen

POCOS – Plain Old Csharp Objects sind einfache C# Klassen, die im Gegensatz zu gutem objektorientierten Stil **nur Zustand (Properties)** anstelle von Zustand und Verhalten (Methoden) enthalten.

Diese **POCOs** stellen am Ende das **Mapping** unser **Tabellen** dar. Jede **Property** matched dabei (tendenziell) ein **Feld** der **Datenbank**.
Später werden wir noch lernen, dass es weitere Möglichkeiten gibt Properties auf den Entitäten vorzuenthalten, obwohl diese als Felder in der Datenbank enthalten sind.

Definitionen auf der anderen Seite, dienen dazu die **objektorientierten Entitäten** und **Tabellen** der **Datenbank** zu **verbinden**. Das **Kernelement** eines **ORM** sozusagen.

Im Video/Cheat-sheet zum Thema „Was ist Entity Framework“ haben wir ja bereits gesehen, dass **Objektorientierung** und **Relationale Daten** sich in ihrer Abhängigkeitsrichtung unterscheiden. Der sogenannte Objekt-Impedanz-Mismatch.

```
Customer(
  Id Integer PK,
  Name Varchar(50),
...
);
```

```
class Customer{
  int Id;
  Order Order;
  string  Name; …
}
```

```
Orders(
  Id Integer PK,
  Amount Integer,
  CustomerId FK,
...
);
```

```
class Order{
  int Id;
  int Amount;
}
```

Abbildung 2 - Objekt - Impedanz Mismatch:
SQL - Order → Customer (FK verweist auf PK)
OO – Customer → Order (Referenz auf Objekt)

Diese **Typdefinitionen** helfen **Entity Framework** dieses **aufzulösen**.

Dazu gibt es mehrere technische Implementierungen:

1. Fluent API
2. Attribute API
3. Konventionen

Mehr dazu im Kapitel über Code first u. Database first approach.

4 ChangeTracking

Das Change Tracking Feature sorgt dafür, dass wir die **CUD** von **CRUD** (Create-Read-Update-Delete) nicht selbst verwalten müssen.
Anders gesagt, Entity Framework **verwaltet** unsere **Updates**, **Inserts** und **Deletes** von Entitäten der Datenbank und wir müssen **lediglich C# code schreiben**.

1. Verwaltet alles in einer Transaktion (mehrere Commands mit einer Session in Folge möglich, nur wenn alle durchgehen, wird dies auch zur Datenbank geschrieben)
2. Sobald wir Daten abfragen, werden diese von EF „getracked“ und das gilt für den gesamten Objektgraph.
3. Das Changetracking ist auch vollständig kontrollierbar.

Das Thema werden wir im Laufe des Kurses sowohl im Kapitel über die EF API als auch im Kapitel zu den fortgeschrittenen Themen betrachten.

5 Code first und Database first

Um den DbContext zu erstellen und die Daten unter Verwaltung von Entity Framework zu bekommen, gibt es zwei Möglichkeiten:

1. Code first
2. Database first

Code first verwenden wir mit den folgenden Schritten:

1. Erstelle POCOs, die die Tabellen repräsentieren
2. Schreibe die Definitionen für die SQL Datentypen und Beziehungen (1-to-1, 1-to-n, n-to-n)
3. Erstelle den DbContext und verweise auf die Entitäten
    a. EntityTypeBuilder, OnConfigure usw.
    b. DbSet<T>
4. Nutze die dotnet CLI zum Erstellen der Datenbank mit einer initialen Migration

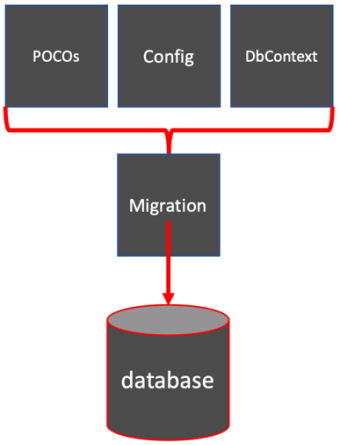

Abbildung 3 - Schematische Darstellung von code-first

Database first verwenden wir mit den folgenden Schritten

1. (Erstellen der Datenbank, eher nutzen einer vorhandenen Datenbank)
2. Nutzen des CLI Tools zum ableiten des DbContext

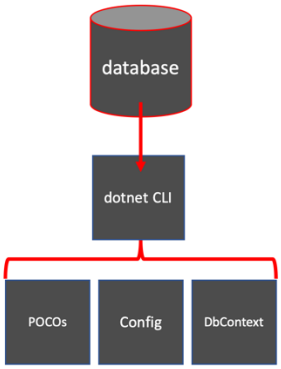

Abbildung 4 - Schematische Darstellung von Database first

Database first oder Code first?

Grundsätzlich gilt, dass Code first zu bevorzugen ist (Single point of truth, initiale Migration). Sollte allerdings eine Legacy DB vorhanden sein, dann sollte Database first verwendet werden.

Zum Lernen von Ef Core ist DbFirst ebenfalls einfacher.

Ausnahmen sind immer zulässig und sollten natürlich im Einzelfall der Domäne, Teamorga usw. betrachtet werden.