

# Entity Framework

...

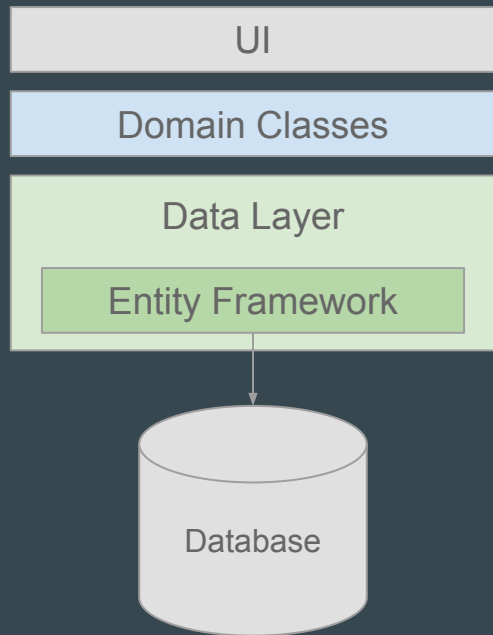
En introduktion

# Entity Framework är:

- Ett open-source ORM framework för .NET applikationer från Microsoft
- Ett system som gör det möjligt att jobba på en högre abstraktionsnivå utan att behöva fundera på den underliggande databasens tables och kolumner

**Official Definition:** “Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.”

# Entity Framework



# Entity Framework Features

- **Cross-platform:** Funkar på Windows, Mac och Linux
- **Modelling:** EF (Entity Framework) creates an EDM (Entity Data Model) based on POCO (Plain Old CLR Object) entities with get/set properties of different data types. It uses this model when querying or saving entity data to the underlying database.
- **Querying:** EF allows us to use LINQ queries to retrieve data from the underlying database. The database provider will translate this LINQ queries to the database-specific query language.
- **Change Tracking:** EF keeps track of changes occurred to instances of your entities (Property values) which need to be submitted to the database.
- **Saving:** EF executes INSERT, UPDATE, and DELETE commands to the database based on the changes occurred to your entities when you call the SaveChanges() method. EF also provides the asynchronous SaveChangesAsync() method.
- **Concurrency:** EF uses Optimistic Concurrency by default to protect overwriting changes made by another user since data was fetched from the database.
- **Transactions:** EF performs automatic transaction management while querying or saving data. It also provides options to customize transaction management.
- **Caching:** EF includes first level of caching out of the box. So, repeated querying will return data from the cache instead of hitting the database.
- **Built-in Conventions:** EF follows conventions over the configuration programming pattern, and includes a set of default rules which automatically configure the EF model.
- **Migrations:** EF provides a set of migration commands that can be executed on the NuGet Package Manager Console or the Command Line Interface to create or manage underlying database Schema.

# Entity Framework - Basic Workflow

1. Definiera en modell. Detta inkluderar att skapa klasser, en context-klass ärvd från DbContext samt eventuella konfigurationer.
2. För att skapa data lägg till ett objekt till din context och anropa SaveChanges()-metoden. EF API bygger ett INSERT-kommando och kör det mot databasen.
3. För att läsa data, använd LINQ-to-Entities queries. EF API konverterar förfrågan till en SQL-query och kör den mot databasen. Resultatet transformeras till ett objekt.
4. För att redigera eller ta bort, uppdatera eller ta bort objekt från en context och kalla på SaveChanges()-metoden. EF API bygger ihop ett passande UPDATE eller DELETE-kommando och kör det mot databasen.

# DbContext

DbContext är den viktigaste klassen när du jobbar med Entity Framework. Den representerar kopplingen till den underliggande databasen mot vilken du utför CRUD-operationer via dess API:

- Databaskopplingar
- Dataoperationer såsom förfrågningar och spara data
- Hålla koll på förändringar
- Model building
- Data Mapping
- Cachning av objekt
- Transaction management

*CRUD* - /krʌd/ - *noun*  
Create, Read, Update, Delete

# Utvecklingsmetoder

Tre olika tillvägagångssätt finns för att jobba med EF:

1. Database-First
2. Code-First
3. Model-First

I Entity Framework Core ligger fokus ofta på Code-First, då det från början varit begränsat stöd för Database-first och inget stöd för Model-first.

# Utvecklingsmetoder

Tre olika tillvägagångssätt finns för att jobba med EF:

1. Database-First  Databasen styr hur koden ska se ut
2. Code-First 
3. Model-First

I Entity Framework Core ligger fokus ofta på Code-First, då det från början varit begränsat stöd för Database-first och inget stöd för Model-first.




# DbSet

`DbSet<TEntity>` representerar en samling av en viss entity i en given model, och fungerar som gränssnittet till databasoperationer med den entityn.

`DbSet<TEntity>` klasser läggs till som properties i `DbContext` och mappas per automatik till tabeller som får samma namn som propertyn.

```
public class MyDbContext : DbContext
{
    public DbSet<Review> Reviews { get; set; }
    public DbSet<Movie> Movies { get; set; }
}
```

Dessa refereras nu till som “entities”



Name	Type	Schema
▼ Tables (4)		
▼ Movies		CREATE TABLE "Movies" ( "M
MovieId	INTEGER	"MovieId" INTEGER NOT NUL
Title	TEXT	"Title" TEXT
Genre	TEXT	"Genre" TEXT
▼ Reviews		CREATE TABLE "Reviews" ( "R
ReviewId	INTEGER	"ReviewId" INTEGER NOT NU
Grade	INTEGER	"Grade" INTEGER NOT NULL
Comment	TEXT	"Comment" TEXT
MovieId	INTEGER	"MovieId" INTEGER

# Vad läggs till databasen?

- `DbSet<TEntity>`
- Refererade klasser i en entity
- Klasser tillagda via `OnModelCreating`

# Blog, Post och AuditEntry läggs alla till databasen

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        b.Entity<AuditEntry>();
    }
}
```

```
public class AuditEntry
{
    public int AuditEntryId { get; set; }
    public string Username { get; set; }
    public string Action { get; set; }
}
```

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}
```

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

# Hur bestäms relationerna?

- **Conventions**

En property som heter Id eller <classname>Id blir primary key osv.

- **Data annotations**

Klasser eller properties som föregås av exempelvis [Key], [Required] osv

- **Fluent API**

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Review>()
        .Property(review => review.Grade)
        .IsRequired();
}
```

Läs mer: <https://docs.microsoft.com/en-us/ef/core/modeling/relationships>

# Hur bestäms relationerna?

- **Conventions**

En property som heter Id eller <classname>Id blir primary key osv.

- **Data annotations**

Klasser eller properties som föregås av exempelvis [Key], [Required] osv

- **Fluent API**

**Har högst prio!**

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Review>()
        .Property(review => review.Grade)
        .IsRequired();
}
```

Läs mer: <https://docs.microsoft.com/en-us/ef/core/modeling/relationships>

**Vi kodar!**

# Sätt upp miljön

- Skapa ett nytt console-projekt:
  - `dotnet new console`
- Installera Entity Framework Core:
  - `dotnet add package Microsoft.EntityFrameworkCore.Sqlite`

# Skapa context och models

- Skapa en mapp för *context*
  - Skapa *MyDbContext.cs*
- Skapa en mapp för *models*
  - Skapa *Movie.cs*
  - Skapa *Studio.cs*




# Skapa databasen

1. `dotnet tool install --global dotnet-ef`
2. `dotnet add package Microsoft.EntityFrameworkCore.Design`
3. `dotnet ef migrations add InitialCreate`
4. `dotnet ef database update`

# Skapa databasen


Extension tool för *dotnet*. Skapar migrationer, utför migreringar, genererar kod för en modell baserad på en existerande databas, med mera.

1. `dotnet tool install --global dotnet-ef` 
2. `dotnet add package Microsoft.EntityFrameworkCore.Design`
3. `dotnet ef migrations add InitialCreate`
4. `dotnet ef database update`


# Skapa databasen

1. `dotnet tool install --global dotnet-ef`
2. `dotnet add package Microsoft.EntityFrameworkCore.Design`
3. `dotnet ef migrations add InitialCreate`
4. `dotnet ef database update`

Extension tool för *dotnet*. Skapar migrationer, utför migreringar, genererar kod för en modell baserad på en existerande databas, med mera.



Används av Entity Framework Core tools



# Skapa databasen

1. `dotnet tool install --global dotnet-ef`
2. `dotnet add package Microsoft.EntityFrameworkCore.Design`
3. `dotnet ef migrations add InitialCreate`
4. `dotnet ef database update`

Extension tool för *dotnet*. Skapar migrationer, utför migreringar, genererar kod för en modell baserad på en existerande databas, med mera.

Används av Entity Framework Core tools

**migrations** skapar en migrationsfil som syftar till att skapa våra första tabeller för vår modell

# Skapa databasen

1. `dotnet tool install --global dotnet-ef`
2. `dotnet add package Microsoft.EntityFrameworkCore.Design`
3. `dotnet ef migrations add InitialCreate`
4. `dotnet ef database update`

Extension tool för *dotnet*. Skapar migrationer, utför migreringar, genererar kod för en modell baserad på en existerande databas, med mera.

Används av Entity Framework Core tools

**migrations** skapar en migrationsfil som syftar till att skapa våra första tabeller för vår modell

**update** skapar vår databas om den inte finns och applicerar vår senast skapade migration

# Inspektera databasen

När du jobbar lokalt med utveckling kan du med fördela använda ett verktyg såsom <https://sqlitebrowser.org/dl/> för att titta på hur din databas ser ut, lägga till data manuellt och göra ändringar.

# Lägga till data

Add<TEntity>(TEntity entity)

```
// med type parameter
var author = new Author{ FirstName = "William", LastName = "Shakespeare" };
context.Add<Author>(author);
context.SaveChanges();
```

```
// utan type parameter
var author = new Author{ FirstName = "William", LastName = "Shakespeare" };
context.Add(author);
context.SaveChanges();
```

# Frågor

- Hur fungerar Cascade delete? Hur kan du göra så att länkade objekt inte tas bort alternativt tas bort?
- Varför är Reviews en ICollection fastän den implementeras som en List?
  - ```
public ICollection<Review> Reviews { get; set; } = new List<Review>();
```



# Bra länkar!

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

<https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>

<https://www.learnentityframeworkcore.com/>

# Movie.cs

```
using System.Collections.Generic;

namespace EFDemo
{
    public class Movie
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Genre { get; set; }
        public ICollection<Review> Reviews { get; set; } = new List<Review>();
    }
}
```

# Review.cs

```
using System.Collections.Generic;

namespace EFDemo
{
    public class Review
    {
        public int Id { get; set; }
        public int Grade { get; set; }
        public string Comment { get; set; }
        public int? MovieId { get; set; }
        public Movie Movie { get; set; }
    }
}
```

# MyDbContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace EFDemo
{
    public class MyDbContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
        public DbSet<Review> Reviews { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=minDatabas.db");
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Review>().Property(r => r.Grade).IsRequired();
        }
    }
}
```

# Program.cs

```
using System;
using System.Linq;

namespace EFDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new MyDbContext())
            {
                //Skapa en ny film
                db.Add( new Movie { Title = "Kill Bill 45", Genre = "Action" } );
                Console.WriteLine("Lade till en film!");
                db.SaveChanges();

                Console.ReadLine();

                //Läs in en film
                var movie = db.Movies
                    .OrderBy(m => m.Id)
                    .First();

                Console.WriteLine("Hämtat ut film: " + movie.Title);

                Console.ReadLine();

                movie.Genre = "Fantasy";
                movie.Reviews.Add( new Review { Grade = 5, Comment = "Helt ok bra film." } );
                db.SaveChanges();

                Console.WriteLine("Satte betyget " + movie.Reviews.FirstOrDefault().Grade + " på " + movie.Title);

                Console.ReadLine();

                Console.WriteLine("Tar bort film: " + movie.Title);
                db.Remove(movie);
                db.SaveChanges();
            }
        }
    }
}
```