Object-Oriented Programming with C#

Database access with Entity Framework Core (EFCore)

By Per Laursen
24-08-2025

Introduktion

Dette dokument indeholder et antal øvelser omhandlende brug af **Entity Framework Core Version 9**, aka **EFCore**-teknologien til at etablere forbindelse mellem en relationel database og en C# konsol-applikation.

Progressionen i øvelserne er som følger:

- **EFCore.0**: Brug af **EFCore** til at tilgå en enkelt tabel
- **EFCore.1**: Brug af **EFCore** til at udføre typiske database-operationer på flere tabeller.
- **EFCore.2**: Brug af et repository-abstraktionslag baseret på **EFCore**.

I alle øvelserne benyttes en domæne-model bestående af entiteterne **Kunde**, **Bil** og **Leje**, hvilket skal modellere f.eks. en virksomhed der udlejer biler.

Alle øvelserne er baseret på *database-first* princippet, hvor man først definerer relationelle tabeller, opretter disse tabeller i en database, og derefter genererer C#-klasser baseret på disse tabel-definitioner. Klasserne genereres ved brug af Visual Studio extension *EFCore Power Tools*.

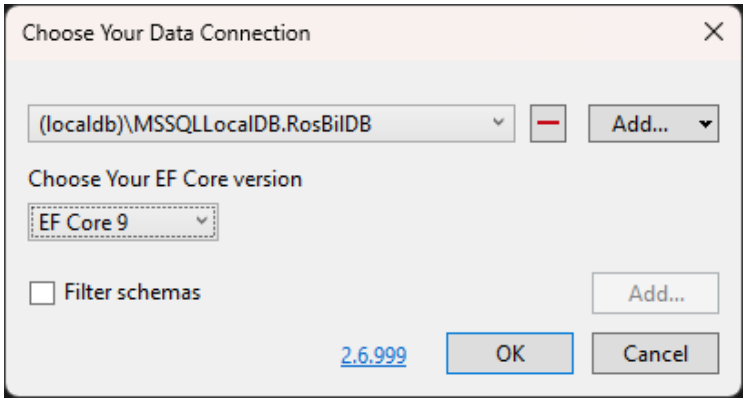
Øvelser

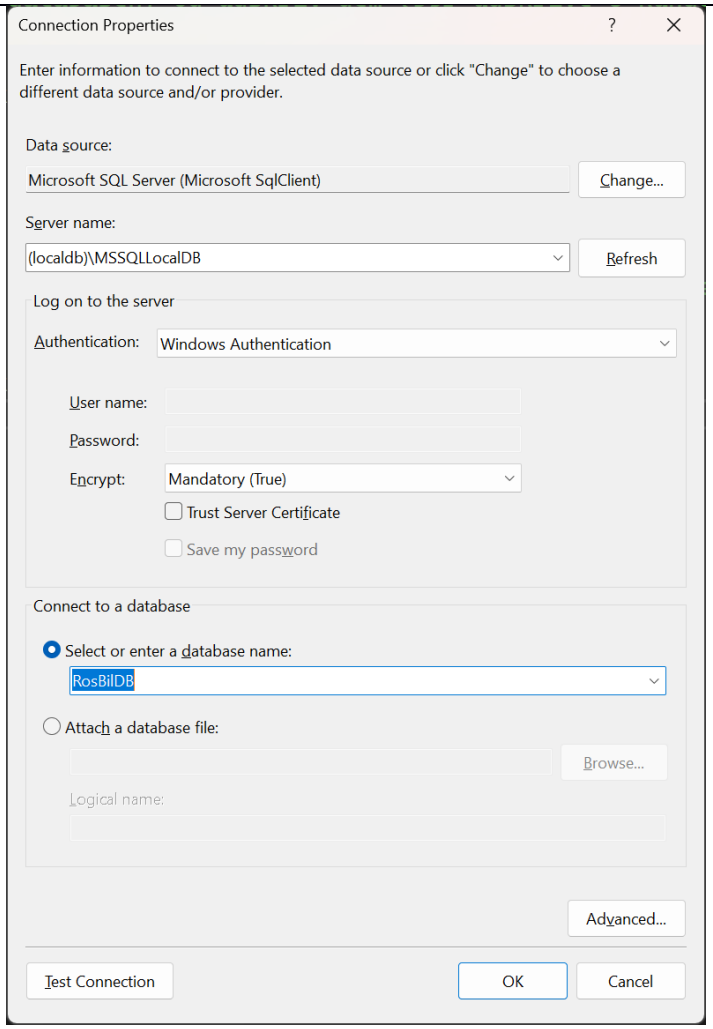
Øvelse	**EFCore.0**
Projekt	**EFCRosBil_V0**
Formål	Brug af **EFCore** og **EFCore Power Tools** til først at tilgå en enkelt tabel, derefter til at generere kode til at tilgå flere tabeller.
Beskrivelse	Projektet rummer: • Klassen **Kunde**, som modellerer kunde-entiteten fra vores domæne-model. • Klassen **RosBilDBContext**, som er en form for *in-memory* repræsentation af databasen (se senere). • Filen **DBScript.sql**, som rummer et SQL script der opretter og populerer tre tabeller i en database (se senere). Bemærk at vi kun benytter tabellen ***Kunde*** i denne øvelse. • Kode i **Program.cs** som læser og processerer data fra tabellen ***Kunde***. Bemærk, at Trin 1+2 kan springes over, hvis du allerede har oprettet **RosBilDB-**databasen
Trin	1. Opret en lokal database kaldet **RosBilDB** (brug **SQL Server Object Explorer** vinduet, gå ned til **SQL Server\(localdb)\MSSQLLocalDB\ Databases**, højre-klik og vælg **Add New Database**). Det er vigtigt at du giver databasen præcist dette navn. 2. Når databasen er blevet oprettet, skal du køre scriptet fra **DBScript.sql** på databasen (højre-klik på databasen, vælg **New Query**. Kopier indholdet af scriptet over i query-vinduet, og kør det). Dette script opretter tre tabeller i databasen. Tjek gerne, at det rent faktisk er sket. 3. Åbn klassen **RosBilDBContext**. Denne klasse er en auto-genereret klasse, som er blevet genereret ved brug af Visual Studio extension *EF Core Power Tools*. Denne klasse repræsenterer (dele af) en database. Det er ikke nødvendigt at forstå alle detaljer i denne klasse, men bemærk at: a. Klassen indeholder en database "connection string", som rummer alle nødvendige detaljer for at kunne forbinde sig til en database. b. Klassen rummer en property **Kunder**, som man kan tænke på som en repræsentation af **Kunde**-tabellen. 4. Fortsæt til **Kunde**-klassen. Dette er en domæne-klasse svarende til entiteten **Kunde** fra vores domæne-model, men denne klasse er også auto-genereret ved brug af *EF Core Power Tools* (dog er **ToString**-metoden tilføjet manuelt). Det er derfor klassen ser lidt atypisk ud, f.eks. med "data annotations" som **[Key]** samt at klassen er erklæret som **partial**. Vi kommer til at prøve at bruge *EF Core Power Tools* senere i denne øvelse. 5. Fortsæt til **Program.cs**. Som det første oprettes et **context**-objekt (med den lidt atypiske **using**-syntaks, som blot sikrer at programmet rydder pænt op efter sig selv, når databasen ikke skal bruges længere). Man kan tænke på dette som at

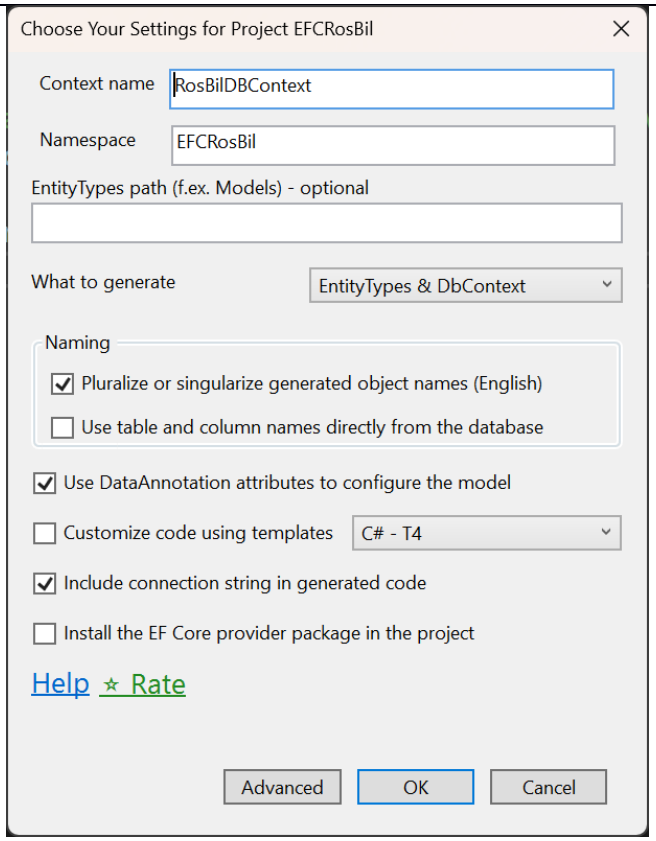
programmet forbinder sig til databasen. Derefter skriver vi alle **Kunde**-objekter ud, ved i **foreach**-loopen at referere til **context.Kunder**. Denne property repræsenterer **Kunde**-tabellen i vores database, så dette er faktisk nok til at data læses ind fra tabellen, omdannes til **Kunde**-objekter og stilles til rådighed i denne property! Detaljerne omkring indlæsning og "oversættelse" mellem tabel-rækker og objekter håndteres af **EFCore**-biblioteket.

6. Den næste **foreach**-loop gør funktionelt præcist det samme, men her refererer vi i stedet til **context.Set<Kunde>()**. Dette kan læses som at vi beder **context**-objektet om alle **Kunde**-objekter der findes i databasen. For nu er den ene måde lige så god som den anden, men den sidste måde kan gøre det nemmere at skrive type-parameteriseret kode, der interagerer med **context**-objektet.
7. Prøv at køre programmet, og se om det forventede data kommer ud på skærmen. Hvis ikke, må du tjekke om du har fået sat databasen korrekt op (se de første trin i øvelsen).
8. Vi skal nu prøve at bruge *EF Core Power Tools* – fremover forkortet *EFCPT* – til at auto-generere mere kode. Dette kræver, at du først installerer denne "extension" til Visual Studio. Første trin er at downloade den fra Visual Studio Marketplace: https://marketplace.visualstudio.com/items?itemName=ErikEJ.EFCorePowerTools
9. Når download er færdig, skal du blot installere den ved følge instruktionerne på skærmen. **NB:** Du skal lukke alle kørende instanser af Visual Studio først! Når installationen er færdig, skal du genåbne **EFCRosBil_V0** projektet. Hvis du nu højre-klikker på projektet (**NB:** Ikke på hele solution, kun projektet!), burde du se et menu-punkt **EF Core Power Tools**.
10. Under **EF Core Power Tools**, vælg menu-punktet **Reverse Engineer**. Nu fremkommer en dialog (se screenshots sidst i øvelsen), hvor du skal tilføje en ny "data connection". Vælg **Add… Database Connection**.
11. I næste dialog skal du i **Server Name** angive navnet på din database-server, som – hvis man kører på en lokal database – vil være **(localdb)\MSSQLLocalDB**. Tast det ind, og test din forbindelse ved at klikke på **Test Connection**. Hvis du efterfølgende folder listen under **Select or enter a database name** ud, bør du kunne se **RosBilDB** i listen. Vælg den, og klik på **OK**.
12. Nu vender du tilbage til den første dialog, hvor der nu skulle være tilføjet en database-connection. Klik på **OK**, og du kommer til dialogen **Choose Your Database Objects**. Her kan du vælge de tabeller, der skal genereres kode for. Da vi allerede har genereret kode for **Kunde**, er den sandsynligvis allerede valgt. I dialogen skal du vælge alle tre tabeller, og klikke **OK**.
13. Nu fremkommer **Choose Your Settings** dialogen. Den bør se ud som på screenshottet, hvis ikke så tilpas den så den gør (**NB**: Første gang skal option *"Install the EF Core provider…"* tilvælges), og klik på **OK**. Et lille "pro-tip": tag en kopi af **ToString**-metoden i **Kunde**-klassen først!
14. Når du klikker **OK**, går auto-genereringen af kode i gang. Dette kan tage nogle få sekunder. Bemærk, at de tidligere auto-genererede klasser bliver overskrevet! Til sidst burde du have fået genereret klasserne **RosBilDBContext**, **Kunde**, **Bil** og **Leje**.
15. Hvis du nu går tilbage til **Program.cs**, vil du se en fejl i den første **foreach**-loop. Slet den, så det kun er den nederste **foreach**-loop der står tilbage (den hvor der refereres til **context.Set<Kunde>()**. Nu bør du kunne køre programmet.

16. Du burde få udskrevet 5 **Kunde**-objekter… men du får sikkert kun udskrevet teksten ***EFCRosBil.Kunde*** for hvert **Kunde**-objekt. Det er fordi **Kunde**-klassen blev overskrevet, da vi auto-genererede kode! Tog du en kopi af **ToString**…? Så tilføj den igen, ellers må du selv skrive en passende **ToString**. Når du er i gang, kan du skrive en **ToString** for **Bil** og **Leje** også. De behøver ikke være helt perfekte fra starten, de kunne f.eks. bare returnere **$"Bil {Id}"** eller noget i den stil.
17. Gå tilbage til **Program.cs**, og lav yderligere to **foreach**-loops, der udskriver alle **Bil**- og **Leje**-objekter. Det kan gøres ved at kopiere den givne **foreach**-loop, og erstatte **context.Set<Kunde>()** med **context.Set<Bil>()** og **context.Set<Leje>()**. Kør nu programmet, du skulle gerne se at alle objekter bliver udskrevet.
18. Åbn **Leje**-klassen. Det er den mest komplekse domæne-klasse, da den har objekt-referencer til **Kunde** og **Bil**. Hvis din **ToString**-metode i **Leje** ikke allerede bruger disse objekt-referencer, omskriv den da så den benytter mindst en af disse, f.eks. således : **return $"Leje {Id} ({Kunde.Navn})"**. Kør programmet; du burde nu se, at denne objekt-reference faktisk bliver sat korrekt i **Leje**.
19. …MEN det er faktisk mere held end forstand, at det "bare virker". Prøv at udkommentere de **foreach**-loops der udskriver **Kunde** og **Bil**, så det kun er **Leje**-objekter der bliver udskrevet. Kør programmet; nu vil du sikkert opleve at få en **NullReferenceException**. Det er fordi objekt-referencerne til **Kunde** og **Bil** kun bliver sat hvis vi allerede har indlæst **Kunde**- og **Bil**-objekterne, hvilket skete da vi havde alle **foreach**-loops kørende. Heldigvis kan vi tvinge **EFCore** til altid at sætte disse objekt-referencer. I **foreach**-loopen, erstat **context.Set<Leje>()** med **context.Set<Leje>().Include(l => l.Bil).Include(l => l.Kunde)**. Dette skal læses som "sæt også objekt-referencerne **Bil** og **Kunde** til at pege på de rigtige **Bil**- og **Kunde**-objekter, også selv om de ikke er indlæst endnu". Kør programmet; nu skulle det gerne virke som forventet.




Connection Properties
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (Microsoft SqlClient)
Change...
Server name:
(localdb)\MSSQLLocalDB
Refresh
Log on to the server
Authentication:
Windows Authentication
User name:
Password:
Encrypt:
Mandatory (True)
Trust Server Certificate
Save my password
Connect to a database
Select or enter a database name:
RosBilDB
Attach a database file:
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel


Choose Your Settings for Project EFCRosBil
Context name
RosBilDBContext
Namespace
EFCRosBil
EntityTypes path (f.ex. Models) - optional
What to generate
EntityTypes & DbContext
Naming
Pluralize or singularize generated object names (English)
Use table and column names directly from the database
Use DataAnnotation attributes to configure the model
Customize code using templates
C# - T4
Include connection string in generated code
Install the EF Core provider package in the project
Help
Rate
Advanced
OK
Cancel

Øvelse	**EFCore.1**
Projekt	**EFCRosBil_V1**
Formål	Brug af **EFCore** til at udføre typiske database-operationer på flere tabeller.
Beskrivelse	Projektet indeholder: • Folderen **Models**, der rummer interfacet **IHarId**, de tre (auto-genererede) domæne-klasser **Kunde**, **Bil** og **Leje**, samt **Extensions.cs**, der rummer tilføjelser til de tre domæne-klasser, i form af partielle klasse-definitioner. • Klassen **RosBilDBContext**, som i store træk er som i den sidste øvelse, dog er der genereret kode til indlæsning af alle tre tabeller. • Klassen **Helpers**, som rummer et par nyttige hjælpe-metoder. • Filen **DBScript.sql**, som er som i den sidste øvelse.
Trin	1. Start i **Extensions.cs**. Denne fil rummer tre partielle klasse-definitioner, der hver især rummer "tilføjelser" til en auto-genereret domæne-klasse. Disse tilføjelser er a. Klassen skal implementere **IHarId** b. En implementation af **ToString** c. En factory-agtig **Create**-metode. Denne metode er defineret, fordi det kan give meget kryptiske problemer med **EFCore**, hvis man definerer parameteriserede constructors for domæne-klasserne… Det er ret teknisk hvorfor dette giver problemer, så det nemmeste er bare at lade være 😊. 2. Fordi vi har implementeret disse "extensions", rummer de andre tre filer kun den auto-genererede kode, så den rører vi ikke ved. Tag dog et kig på klassen **Leje**. Bemærk, at denne klasse både har properties som er objekt-referencer af type **Bil** og **Kunde**, men også properties svarende til fremmed-nøglerne i tabellen…? Hvorfor dog begge dele? Det kommer i spil, når man gerne vil oprette nye **Leje**-objekter, og efterfølgende gemme dem i databasen. Hvis man opretter et nyt **Leje**-objekt, der skal referere til eksisterende **Bil**- og **Kunde**-objekter, skal dette gøres ved at sætte de properties der svarer til fremmednøglerne, og kun dem! Hvis man sætter objekt-referencerne direkte, vil de af **EFCore** blive opfattet som nye objekter, og man vil derfor få en fejl, når **EFCore** prøver at gemme disse i databasen, da der jo allerede findes f.eks. en **Kunde**-række med den primære nøgle, som svarer til **Id** på **Kunde**-objektet. Der er også derfor, at **Create**-metoden for **Leje**-klassen kun tager id'er som parametre, ikke objekter! 3. Fortsæt til **Program.cs**. Her ses eksempler på hvordan man i praksis bruger **EFCore** til at læse, oprette og slette data. Nogle vigtige pointer:

a. I trin 2 (og 4, 6, 8, og 10) skal vi læse data fra databasen, ved brug af referencerne til f.eks. **context.Set<Bil>()**. Bemærk, at koden er lidt mere kompleks når vi skal indlæse **Leje**-objekter, da disse jo har objekt-referencer. Det var det vi eksperimenterede med i den sidste øvelse; brug af **Include** sikrer, at disse objekt-referencer bliver sat korrekt.
b. I trin 3 og 7 skal vi oprette nye objekter. Her er det stadig vores ansvar at finde **Id**'er for de nye objekter, derfor kaldene til **Helpers.FindMaxId**. Når de nye objekter er klar, kaldes **context.Set<Kunde>().Add(k1)** (for **Kunde**-objekter). Bemærk, at objekter dog først bliver endeligt tilføjet til databasen, når vi kalder **context.SaveChanges()**. Det er også vigtigt at bemærke, at det ikke virker at tilføje det nye objekt til den **List**, som bliver dannet ved at kalde **context.Set<Kunde>().ToList()**, da kaldet af **ToList** laver en liste som ikke længere er forbundet til databasen!
c. I trin 5 og 9 skal vi slette eksisterende objekter. Dette gøres med **Remove**-metoden: **context.Set<Kunde>().Remove(k1)**. Bemærk, at parameteren til **Remove** ikke er et **Id**, men derimod hele objektet der skal slettes. Også her skal man huske at kalde **SaveChanges**, da ændringerne ikke bliver udført før metoden kaldes.

4. Hvis du har tid tilbage, kan du prøve at eksperimentere i **Program.cs** med at oprette, læse og slette flere data. Du kan f.eks. også prøve at skrive kode som returnerer data der opfylder en given betingelse (f.eks. alle VIP-kunder).

Øvelse	**EFCore.2**
Project	**EFCRosBil_V2**
Formål	Brug af et **repository**-abstraktionslag baseret på **EFCore**.
Beskrivelse	Projektet indeholder de samme elementer som i den sidste øvelse, samt disse tilføjelser: • Folderen **Repositories**, som indeholder interfacet **IRepository**, base-klassen **EFCRepositoryBase** samt tre sub-klasser. • Folderen **UI**, som simulerer en UI-struktur der kan minde om en – meget simplificeret – *Razor Pages* app • Klassen **DataService**, som giver en samlet adgang til alle data i databasen.
Trin	1. Start i folderen **Repositories**, med interfacet **IRepository**. Er der noget i definitionen af **IRepository**, der indikerer hvordan data er opbevaret? 2. Fortsæt til **EFCRepositoryBase**. Denne klasses formål er at bruge **EFCore** til at implementere **IRepository**. Sørg for at opnå en rimelig forståelse af, hvordan de enkelte metoder er implementeret. 3. Fortsæt til de tre **…Repository** klasser. De ser nærmest trivielle ud. Hvad er deres formål? Kunne de måske undværes? Og hvorfor er **LejeRepository** lidt anderledess end de to andre? 4. Fortsæt til **DataService**. Hvad er denne klasses formål? Hvorfor har de tre properties en interface-type i stedet for konkrete …**Repository**-typer? 5. Fortsæt til **Program.cs**. Strukturen skulle gerne minde om strukturen af **Program.cs** i den sidste øvelse, men fremgår det nu af koden i **Program.cs**, at vi bruger **EFCore**…? Kan man ud fra koden konkludere, at data må ligge i en database? Hvilke fordele kan det give? 6. Fortsæt til folderen **UI**. Denne del af programmet har ikke som sådan noget med databaser at gøre, men prøver at simulere en UI-struktur som minder om strukturen i en *Razor Pages* app. Udforsk klasserne, og find ud af hvordan UI og data er koblet sammen. 7. Eksperimentér gerne i **Program.cs** med at bruge mulighederne for at oprette, læse og gemme data. Hvis du har mere tid, kan du prøve at oprette en helt ny tabel (f.eks. en **Medarbejder**-tabel), og se hvor meget/lidt det kræver at implementere hele "kæden" fra data i databasen til at se data på skærmen.