|  |
| --- |
|  |
| Object-Oriented Pro-gramming with C# |
| Database access with ADO.Net |

|  |
| --- |
| By Per Laursen  24-08-2025 |

[Introduktion 2](#_Toc206947306)

[Øvelser 3](#_Toc206947307)

[ADONet.0 3](#_Toc206947308)

[ADONet.1 5](#_Toc206947309)

[ADONet.2 6](#_Toc206947310)

[ADONet.3 8](#_Toc206947311)

# Introduktion

Dette dokument indeholder et antal øvelser omhandlende brug af **ADO.Net**-tekno­logien til at etablere forbindelse mellem en relationel database og en C# konsol-applikation.

Progressionen i øvelserne er som følger:

* **ADONet.0**: Brug af **ADO.Net** til at tilgå en enkelt tabel
* **ADONet.1**: Brug af **ADO.Net** til at tilgå en enkelt tabel. Brug af en dedikeret klasse som rummer kode til typiske database-operationer (læs, slet, opret)
* **ADONet.2**: Brug af **ADO.Net** til at tilgå flere relaterede tabeller. Brug af et mere komplekst sæt af klasser til at udføre typiske database-operationer (læs, slet, opret)
* **ADONet.3**: Brug af **ADO.Net** til at tilgå flere relaterede tabeller. Brug af et **repository**-abstraktionslag.

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

# Øvelser

|  |  |
| --- | --- |
| **Øvelse** | ADONet.0 |
| **Projekt** | **ADORosBil\_V0** |
| **Formål** | Brug af **ADO.Net** til at tilgå en enkelt tabel |
| **Beskrivelse** | Projektet rummer:   * Klassen **Kunde**, som modellerer kunde-entiteten fra vores domæne-model. * 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***.   **NB:** Dette projekt bruger NuGet-pakken *Microsoft.Data.Sql­­Client*, som instal­leres første gang man åbner projektet. Dette kan godt tage nogle sekunder, hvor det vil ligne at der er fejl I programmet. Hvis pakken ikke bliver installeret automa­tisk, kan man i **Program.cs** holde muse-cursoren over f.eks. klassen **SQLConnection**. Dette bør få Visual Studio til at vise dig en mulighed for at installere pakken, som vist nedenfor.  Et billede, der indeholder tekst  Automatisk genereret beskrivelse  (øvelsen fortsætter på næste side) |

|  |  |
| --- | --- |
| **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. Fortsæt til **Program.cs**. Her er kode som benytter **ADO.Net**-biblioteket til at læse data fra databasen. 4. Trin 1 er opsætning af database-forbindelsen, ved brug af klassen **SqlCon­nectionStringBuilder**. Denne klasse gør det nemmere at konstruere den “connection string”, som bruges til at oprette forbindelse til data­basen. 5. Trin 2 er selve brugen af data fra databasen. Dette består af tre del-trin:    1. Trin 2a etablerer en forbindelse til databasen ved brug af klassen **SqlConnection**. Bemærk den lidt usædvanlige brug af **using**; dette sikrer, at forbindelsen til databasen bliver frigivet, selv om der skulle gå noget galt under arbejdet med databasen.    2. Trin 2b definerer og udfører den SQL statement, vi gerne vil udføre på databasen, ved hjælp af klasserne **SqlCommand** (hvor vi define­rer vores SQL statement) og **SqlDataReader** (hvor vi udfører vores SQL statement).    3. Trin 2c processerer det resultat vi fik tilbage ved at udføre vores SQL statement. Dette resultat kommer i form af et **SqlData­Reader**-objekt, på hvilket vi kan læse hver række af data en efter en. 6. 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). 7. Brug også lidt tid på at blive fortrolig med, hvad der foregår under hele arbejdet med databasen. Hvilke klasser har ansvar for hvad? Hvordan kommer vi fra en række i **Kunde**-tabellen til et nyt **Kunde**-objekt? 8. Når du er rimeligt fortrolig med koden, kan du prøve at skrive kode som på tilsvarende måde læser data fra **Bil**-tabellen. Som en del af dette skal du også definere en **Bil**-klasse. 9. Hvis du også er hurtigt færdig med at læse data fra **Bil**-tabellen, kan du også prøve at læse data fra **Leje**-tabellen. Her støder du nok på nogle udfordringer, i forhold til hvordan data i denne tabel refererer til data i de andre tabeller. Er det lige så nemt at lave et **Leje**-objekt som det var at lave et **Kunde**- eller **Bil**-objekt…? |

|  |  |
| --- | --- |
| **Øvelse** | ADONet.1 |
| **Projekt** | **ADORosBil\_V1** |
| **Formål** | Brug af **ADO.Net** til at tilgå en enkelt tabel. Brug af en dedikeret klasse som rummer kode til typiske database-operationer (læs, slet, opret) |
| **Beskrivelse** | Projektet rummer:   * Klassen **Kunde**, som er som i den sidste øvelse. * Klassen **DBMethodsKunde**, som indeholder et antal metoder til at udføre typiske database-operationer på **Kunde**-tabellen, såsom at læse, oprette og slette data (vi udelader muligheden for at opdatere eksisterende data). Metoderne benytter klasserne fra **ADO.Net**-biblioteket. * Klassen **Helpers**, som rummer et par nyttige hjælpe-metoder. * Filen **DBScript.sql**, som er som i den sidste øvelse. |
| **Trin** | 1. Åbn **Program.cs**. Vi starter med at sætte database-forbindelsen op, på samme måde som i sidste øvelse. I trin 2 til 6 bruger vi nu metoderne i **DBMethodsKunde** til at arbejde med data i **Kunde**-tabellen. Start med at se koden (og de tilhørende kommentarer) i **Program.cs** igennem, kør pro­grammet, og se om output er som du ville forvente det. 2. Fortsæt i klasssen **DBMethodsKunde**. Den rummer de tre **public** metoder **ReadAllFromDB**, **WriteToDB** og **Delete­FromDB**. Hovedformålet med opga­ven er at forstå hvordan disse tre metoder fungerer, så brug noget tid på grundigt at studere disse tre metoder. Det inkluderer også at studere de **private** hjælpe-metoder i klassen. Overvej også følgende:    1. Alle metoderne er skrevet specifikt til at arbejde med **Kunde**-tabel­len. Hvor meget vil det kræve at gøre metoderne generelt anven­de­lige på alle tabeller i en database?    2. Lige nu sætter vi **Id** for et nyt **Kunde**-objekt som en del af **Kunde**-constuctoren. Kunne ansvaret for at finde det næste ledige **Id** (vi antag­er at alle **Kunde**-objekter skal have et unikt **Id**) flyttes til **WriteToDB**-metoden? 3. Når du føler du har en god forståelse af metoderne i **DBMethodsKunde**, kan du prøve at lave en tilsvarende klasse for **Bil**-klassen, og udføre nogle operationer der svarer til dem der udføres for **Kunde** i **Program.cs**. Når det virker, kan du – igen – reflektere lidt over, hvad der skal til for at skrive en generelt anvendelig **DBMethods** klasse. Prøv også gerne at få hul på at implementere klassen 😊. |

|  |  |
| --- | --- |
| **Øvelse** | ADONet.2 |
| **Projekt** | **ADORosBil\_V2** |
| **Formål** | Brug af **ADO.Net** til at tilgå flere relaterede tabeller. Brug af et mere komplekst sæt af klasser til at udføre typiske database-operationer (læs, slet, opret) |
| **Beskrivelse** | Projektet indeholder:   * Folderen **Models**, med det meget lille interface **IHarId**, og de tre domæne-klasser **Kunde**, **Bil** og **Leje**. Bemærk at **Leje** rummer to properties som har en klasse-type, dvs. et **Leje**-objekt vil have en reference til et **Kunde**-objekt og et **Bil**-objekt. * Folderen **DBMethods**, som rummer base-klassen **DBMethodsBase**, samt flere sub-klasser svarende til de tre domæne-klasser. Bemærk, at der er to sub-klasser (**DBMethodsLeje** og **DBMethodsLejeJoin**) svarende til en domæne-klasse **Leje**. * Klassen **Helpers**, som rummer et par nyttige hjælpe-metoder. Bemærk, at nogle af disse nu er type-parameteriserede. * Filen **DBScript.sql**, som er som i den sidste øvelse.   Vi har nu et mere komplekst system af klasser, men koden til at læse data fra data­basen og “oversætte” disse data til objekter ligger stadig i **DBMethods**-klasserne. Det der komplicerer tingene er at klassen **Leje** har to objekt-referencer, som vi skal etablere ud fra de fremmed-nøgler der findes i **Leje**-tabellen. |
| **Trin** | 1. Start i **DBMethodsBase**-klassen. Denne klasse er en generalisering af den **Kunde**-specifikke **DBMethodsKunde**-klasse fra sidste øvelse. Der bruges flere forskellige mekanismer til at opnå denne generalisering. Da denne klasse er central for øvelsen, skal du nu bruge noget tid på at studere den grundigt, inklusive at prøve at forstå de mekanismer der er brugt for at gøre klassen så generel som muligt. 2. Fortsæt til **DBMethodsKunde** og -**Bil**. Hvad er det der gør, at disse to klasser bliver specifikke for en enkelt tabel? Hvor er det, at vi “oversætter” fra f.eks. en **Kunde**-række til et **Kunde**-objekt? Og hvorfor er det, at den “oversættelse” ikke er så kompliceret for disse to klasser. 3. Fortsæt til **DBMethodsLeje**. Her støder vi på problemet med at komme fra fremmed-nøgler til objekter (og vice versa). Dette problem skal løses i metoderne **GetRow** (fra fremmed-nøgler til objekter) og **AddParameter­Values** (fra objekt til fremmed-nøgler). M.h.t. **GetRow** ser det ud til, at vi gør brug af referencer til de andre **DBMethods**…-klasser. Hvorfor er det nødvendigt? Hvor bliver disse referencer etableret? 4. Fortsæt til **Program.cs**. Her bliver de nye klasser taget i brug. Arbejd dig igennem de enkelte trin, indtil du synes du har en rimelig forståelse af hvordan klasserne bruges. Kør programmet for at se hvad der sker; det skulle gerne være sådan, at programmet efterlader databasen i den samme tilstand som da programmet startede. Bemærk i øvrigt, at vi nu blot angiver “0” (nul) som **id** når vi opretter et nyt objekt. Hvorfor går det godt? Hvem er det, der nu har ansvar for at finde et korrekt **Id** for et nyt objekt? 5. Fortsæt til **DBMethodsLejeJoin**. Denne klasse benytter en anden tilgang til at konstruere **Leje**-objekter. Mere specifikt bruges en mere kompleks, *join*-baseret SQL statement til at læse det relevante data. Prøv evt. først at køre denne SQL statement direkte på databasen i et query-vindue, for at få en fornemmelse af det data den returnerer. Tag derefter et grundigt kig på metoden **GetRow**, så du forstår hvorfor vi har information nok til at lave et komplet **Leje**-objekt, inklusive objekt-referencer til de korrekte **Kunde**- og **Bil**-objekter. Sammenlign gerne med **DBMethodsLeje**. Hvilke fordele og ulemper kan du se i de to tilgange til at løse problemet? 6. Gå tilbage til **Program.cs**, og prøv at bruge **DBMethodsLejeJoin** i stedet for **DBMethodsLeje**. Det skulle helst ikke gøre nogen forskel 😊. 7. Hvis du har tid tilbage, kan du prøve at eksperimentere i **Program.cs** med at oprette, læse og slette flere data. Du kan også prøve at tilføje metoder til **DBMethods**-klasserne, f.eks. metoder som returnerer data der opfylder en betingelse (f.eks. alle VIP-kunder). |

|  |  |
| --- | --- |
| **Øvelse** | ADONet.3 |
| **Projekt** | **ADORosBil\_V3** |
| **Formål** | Brug af **ADO.Net** til at tilgå flere relaterede tabeller.  Brug af et **repository**-abstraktionslag. |
| **Beskrivelse** | Projektet indeholder de samme elementer som i den sidste øvelse, samt disse tilføjelser:   * Folderen **Repositories**, som indeholder interfacet **IRepository**, base-klassen **ADORepositoryBase** 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 **ADORepositoryBase**. Denne klasses formål er at bruge de eksisterende **DBMethods…** klasser 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? 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 **ADO.Net**…? Kan man ud fra koden konkludere, at data ligger 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. |