Repository med MVC

Skrevet af Henrik Obsen

1. udgave december 2015

Indhold

Indledning ... 3

Om materialet ... 3

Duser kort fortalt ... 4

Databasen ... 4

Opsætning af Websitet AutoKurt ... 5

Hvad er et repository ... 7

Opsætning af repositoryet ... 8

Brug at autofunktionerne ... 9

Hvad er en model? ... 10

Hvad er en Factory? ... 11

Brug af autofunktionen Get() ... 12

Brug af autofunktionen GetAll() ... 14

Bil modellen og BilFac ... 15

Brug af autofunktionen GetBy() ... 17

Brug af autofunktionen Insert() ... 20

Brug af autofunktionen Update() ... 21

Brug af autofunktionen Delete() ... 22

Foto Bloggen ... 24

Byg dine egne funktioner ... 25

Mapping ... 30

Manuel mapping ... 31

Nyttige funktioner ... 32

Random ... 32

Count ... 32

Update enkelte felter ... 34

Opret og retuner ID ... 35

Slet fra flere tabeller ... 36

Hent data fra flere tabeller ... 37

Billede modellen og BilledeFac ... 37

ViewModel BilMedBilleder ... 37

Udtræk af en bil med billeder ... 39

Udtræk af flere bil med billeder ... 41

Inner Join ... 44

En Avanceret søgefunktion ... 46

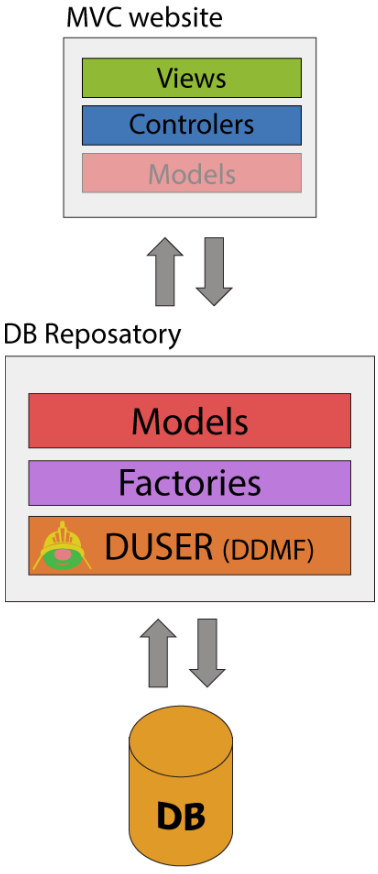
Brugersystemet ... 50

Bruger og BrugerFac 50
Login 51
Opbygning af DDMF 55
Conn.cs 55
Mapper.cs 56
AutoFac 59

Indledning

Programmering af hjemmesider, der benytter en database kan indebære meget kompleks kode. Jo mere kompleks koden bliver, jo sværere bliver det at rette og vedligeholde. Et Repository design pattern er en måde at indføre arkitektur i din kode og skaber en klar lag adskillelse mellem din brugergrænseflade (Views, Controllers, Delegates og håndtering af andre grænsefladespecifikke opgaver), forretningslogik(Class library med alt der omhandler database kommunikation og ikke andet) og til sidst database. Et Repository Pattern kan hjælpe til en mere overskuelig arkitektur og lave nogle mere løst koblede klasser, der nemmere kan genbruges i fremtidige projekter.

Et eksempel kunne være modellen på billedet her til højre. Her har vi en MVC Website der indeholder vores brugerflade specifikke ting og under det et database repository med de lag der skal bruges for at arbejde med vores database. Mere snak om det senere.



Om materialet

Via eksemplerne videre i materialet vil jeg prøve at forklarer helt fra bunden, hvordan vi kan opbygge et database repository. Vi starter helt fra bunden af og ser på hvordan du nemt kan komme i gang med at bruge nogle automatiske funktioner der kan lette dit arbejde. Vi prøver også at lave nogle meget manuelle funktioner for at du kan få forståelsen for hvordan det hele virker. Vi skulle gerne ende ud med, at du kan bygge dit eget repository med auto funktioner der ikke er afhængigt af hverken Modeller eller databasen.

Du kommer også til at gennemgå en række små cases, hvor vi kommer omkring flere af de almindelige opgaver, der kan være i at bygge et website op fra bunden.

Her i materialet opbygger vi vores repository med en data mapper kaldet Duser. Duser har en række grundlæggende metoder som du ikke er van til at bruge, hvis du tidligere har arbejdet med et disconnectet DataAccess-lag der via en dataadapter lægger data fra en database ud i en DataTable.

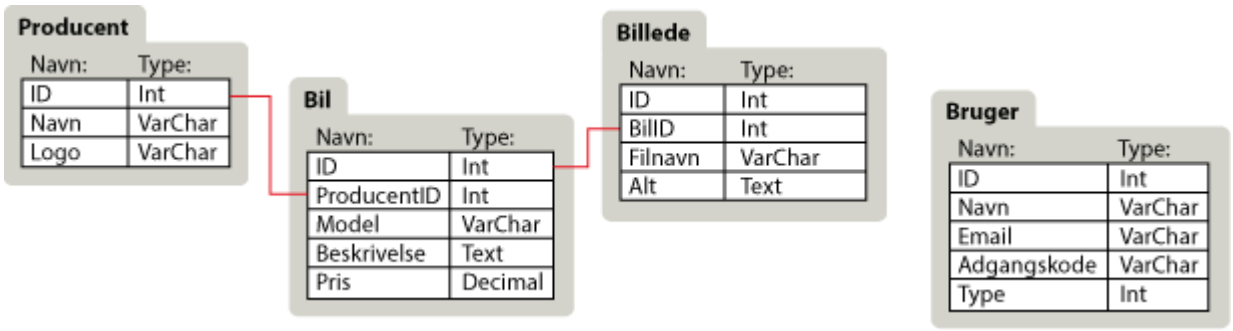
Duser kort fortalt

Duser er kort sagt en folder med 3 klasse-filer der indeholder Data-mapping logik der gør det nemmere for dig at komme i gang med at bruge databaser i dine projekter. Duser findes i forskellige udgaver men vi starter med den helt skrabet udgave og som vi selv bygger videre på.

Alle eksemplerne i materialet vil bygge på et lille website jeg har valgt at kalde Auto Kurt.

Databasen

Jeg har bygget en lille database op til Auto Kurts hjemmeside. Den består af nogle relaterede tabeller der blandt andet indeholder de biler Auto Kurt sælger.



Producent: Indeholder de to bilproducenter som der forhandles (Opel og Chevrolet)
Bil: Indeholder data på de biler der er til salg.
Billede: Indeholder et eller flere billeder af hver bil.
Bruger: Skal bruges til login. Indeholder bla. hashed passwords.

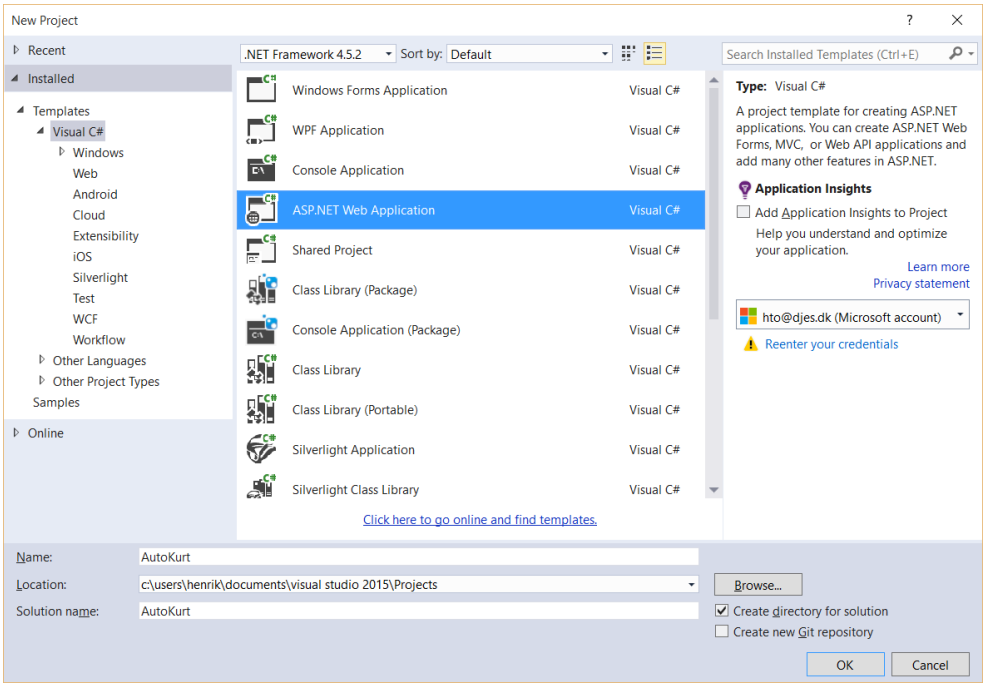
Jeg har lavet databasen på forhånd så vi ikke skal bruge tid på det. Databasen er en MS SQL database. Login oplysningerne finder du her under.

Host: 194.255.108.50
Database: dbAutoKurt
Username: AutoKurt
Password: eG8rYyC3

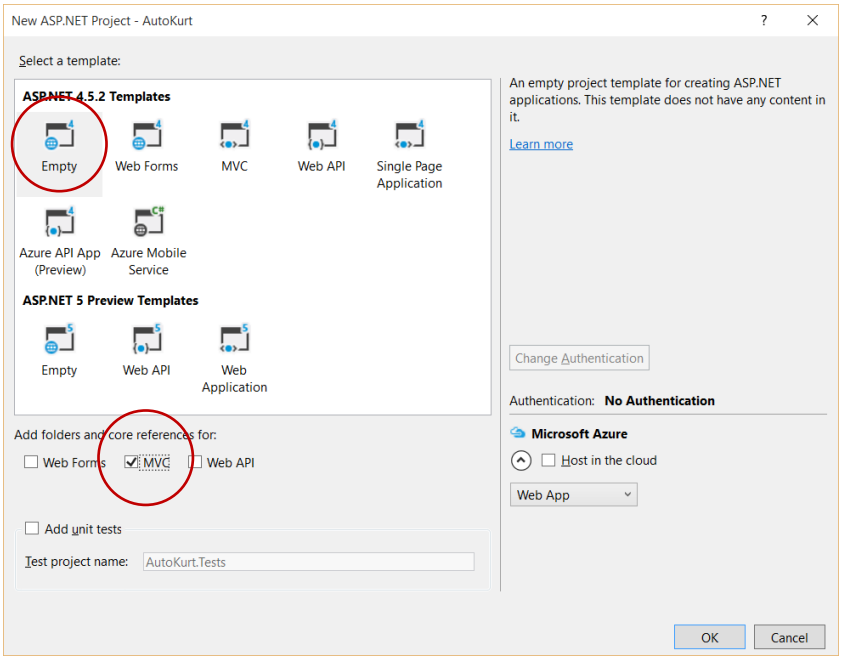
Opsætning af Websitet AutoKurt

Inden vi går i gang med at opbygge et repository skal vi lige have oprettet et nyt website til projektet.

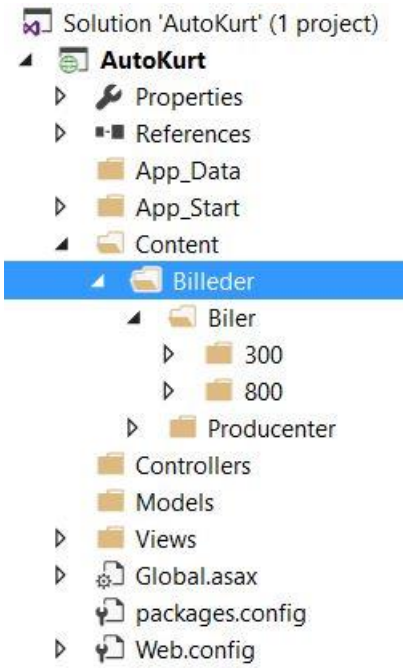
1) Opret et nyt ASP.NET Web Application i Visual Studio. Kald det for AutoKurt. Sørg for at sætte dig fast på hvilken version af .net du vil køre så både dit website og repository kommer til at køre samme version.



2) Vælg Empty og sæt hak i MVC som vist på billedet her under.



3) Opret en ny mappe i roden af dit websted, Kald den ”Content”. Tilføj derefter mappen Billeder til Content som vist på billedet her. Du kan finde den under Downloads på Duser.net. http://www.duser.net/Download/Helpers/Billeder.rar



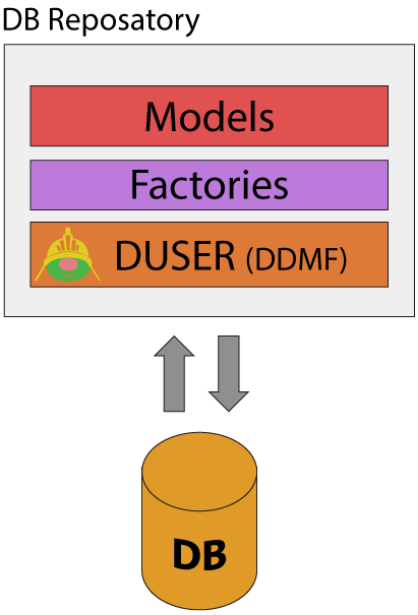
Dit website skulle nu gerne se ud som på billedet her. Vi har en Solution der hedder AutoKurt hvor i der ligger et website med samme navn.

Vi skal i næste skridt have oprettet vores repository i den nye Solution og have lavet en reference mellem websitet og repository.

Hvad er et repository

Et database repository er en applikation der har til formål at kommunikere med vores datakilde, i dette til fælde vores MS SQL Server. I vores repository som består af et Class Library skriver vi alt det kode der skal bruges for at kommunikere med databasen. Hvis du har arbejdet med databaser og programmering før har du sikkert prøvet at lave Property og factory klasser. Det er blandt andet disse klasser der vil være i vores repository.

Der må kun være klasser der i som bruges til database kommunikationen. Prøv at holde det lidt adskilt så dine klasser for eksempel til upload af filer eller afsendelse af mails stadig ligger i dit MVC Website og ikke i dit repository.



Når nu vi starter får du en mappe med nogle hjælpe klasser i. Det er det lag der hedder Duser som du kan se på billedet. Med Duser kan du lave 80% af det database kommunikation som man normalt vil lave, næsten uden at kode. Vi ser senere på hvordan du udvider og selv laver disse hjælpeklasser men det er en nem måde at komme i gang på.

Som nævnt tidligere findes der flere udgaver af Duser DDMF. Den udgave vi bruger er en skrabet udgave vi selv kan bygge videre på. Du kan downloade DDMF Light her: http://www.duser.net/download/DDMF/DDMF_Light.rar

Laget Models inde holder modeller og view models der definer datagrundlaget som, du arbejder med i en Factory, Tabel eller i et View mm. Modellerne et typisk en afspejling af en tabel i databasen tilføjet nogle Data Annotations.

Factories indeholder klasser der behandler det data som kommer fra databasen inden det bliver sendt videre i en Model til en controller eller lignende.

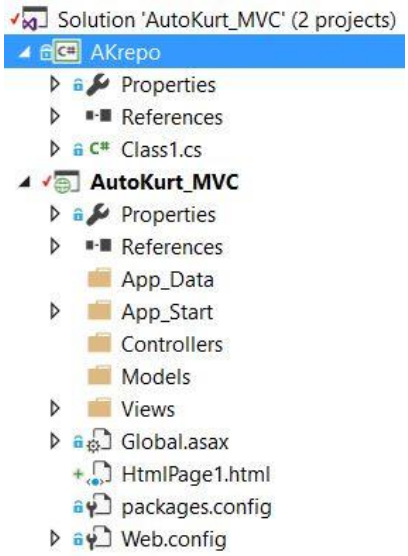
Opsætning af repositoryet

Vi skal have lavet et nyt repository. Som jeg nævnte tidligere er det bare et Class Liberary med en række hjælpeklasser til at kommunikere med vores database.

1) Højre klik på din solution (AutoKurt) i din Solution Explorer og vælg ”Add new project”.

2) Vælg et Class Library og kald det ” AKrepo”.

3) Sikre dig at AKrepo kører samme version af .net som dit website



Det skulle gerne ligne billedet her ude til højre. Før vi kan gå videre skal vi lige have oprettet en reference mellem websitet og AKrepo. Dette gøres på følgende måde.

1) Højreklik på websitet AutoKurt.

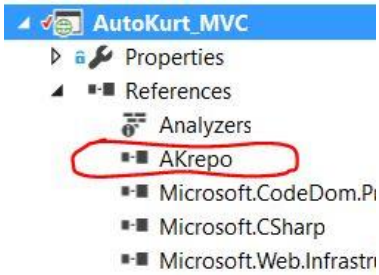
2) Vælg Add Reference.

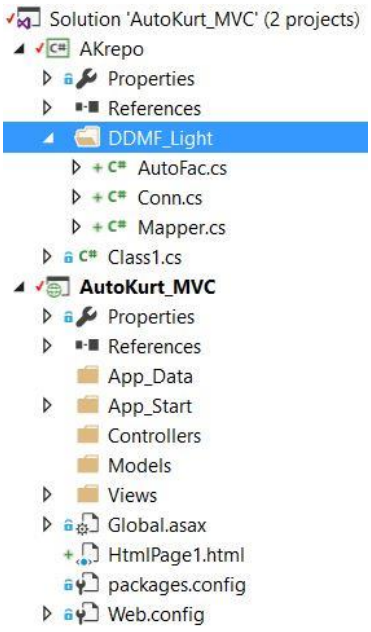
3) Under Project, sætter du flueben ud for AKrepo og klikker på OK.

Nu har vi lavet en reference mellem de to projekter men lad os lige tjekke om det hele virker som det skal.

1) Gå i menuen Build og vælg Build Solution.

2) Dit repository (AKrepo) skulle nu gerne være buildet og lagt i din referance-mappe på dit website. Se billedet her.





Nu skal vi have tilføjet Duser til vores repository. Download og udpak DDMF Light og tilføj mappen til dit class liberary.

Hvis du ikke har downloadet det så kan du hente det her: http://www.duser.net/download/DDMF/DDMF_Light.rar

Tilføj mappen DDMF_Light til AKrepo

Vi skal lige have rettet navnet på Namespacet i de 3 klasser i mappen DDMF_Light

1) Åben filen AutoFac.cs, ret navnet ”REPO_NAMESPACE” til ”AKrepo”. Gem og luk filen igen.

2) Åben filen Mapper.cs, ret navnet ”REPO_NAMESPACE” til ”AKrepo”. Gem og luk filen igen.

3) Filen Conn.cs, ret navnet ”REPO_NAMESPACE” til ”AKrepo”. Ret databaseforbindelsen til med følgende oplysninger. Gem og luk filen igen.

Host: 194.255.108.50
Database: dbAutoKurt
Username: AutoKurt
Password: eG8rYyC3

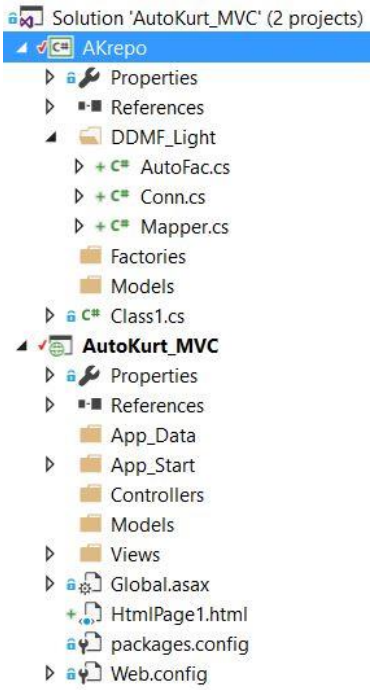
4) Kør et build.

Hvis du ikke får en fejl, bør alt fungere som det skal. Det sidste vi mangler før vi er klar til at kode er to mapper.

1) Opret en mappe med navnet Models i AKrepo.

2) Opret en mappe med navnet Factories.

Dit projekt skulle nu gerne se ud som på billedet her og vi er ved at være klar til at kode. Her under er en kort forklaring af de tre mapper i dit repository.



Models: Bruges til alle dine Modeller, du har nok tidligere hørt dem omtalt som property-klasser men her kalder vi dem for Modeller.

Factories: Denne mappe indeholder alle dine factory-klasser som du nok kender fra tidligere.

DDMF_Light: Indeholder de hjælpeklasser som vi bruger for ikke at skulle kode det samme funktionalitet igen og igen.

Lad os lige se lidt på de filer som ligger i mappen DDMF. Dette er kun en kort gennemgang, vi går i dybden med koden bag det hele senere.

Conn.cs
Denne fil indeholder selve forbindelsen til din database. Det er her i du vil skulle ændre din database-forbindelse når det bliver nødvendigt. Indeholder metoden CreateConnection()

Mapper.cs
Mapperen bruges til at mappe dit indhold fra dine tabeller over i Modeller og Lister. Indeholder metoderne Map() og MapListe().

AutoFac.cs
Denne fil indeholder alle vores automatiske funktioner der opføre sig, forskelligt alt efter, hvilken Model der bliver puttet i den. Indeholder, Get(), GetAll(int ID), GetBy(string feldt, int value), Insert(Model), Update(Model), Delete(int ID).

Du lærer selv at skrive dine egne hjælpeklasser senere i materialet.

Brug at autofunktionerne

Inden vi kan komme i gang med at bruge vores autofunktioner på en tabel er der lige et par småting der skal gøres. Vi skal have oprettet en model til en tabel og vi skal have lavet en factory.

Som udgangspunkt skal du have en model og en factory for hver tabel du har i databasen, dog med få undtagelser.

Hvad er en model?

En model er en direkte afspejling af en tabel (se billedet), grunden til dette er at vores hjælpeklasser bruger modellen til at finde det rigtige data i databasen. Billederne her under viser model Producent og Tabellen Producent hvor navne og datatyper passer sammen.

```
1  namespace AKrepo
2  {
3      public class Producent
4      {
5          public int ID { get; set; }
6          public string Navn { get; set; }
7          public string Logo { get; set; }
8      }
9  }
```

Producent

Navn:	Type:
ID	Int
Navn	VarChar
Logo	VarChar

Inden vi går videre skal vi lige have oprettet model med navnet Producent.

1) Opret en ny klasse i mappen Models. Kald den for Producent lige som tabellen i databasen

2) Ret linjen med class Producent til så den bliver public som vist på billedet her over.

3) Tilføj de tre properties som vist på billedet.

4) Tjek at dit namespace hedder AKrepo og ikke AKrepo.Models som på billedet.

Hvad er en Factory?

Factories er heller ikke helt som det du har prøvet tidligere. Princippet er dog det samme men vores factory her, nedarver fra en af vores hjælpeklasser (AutoFac) som gør at den har en lang række funktioner som du normalt selv ville sidde og kode. Se forskellen på billederne her under.

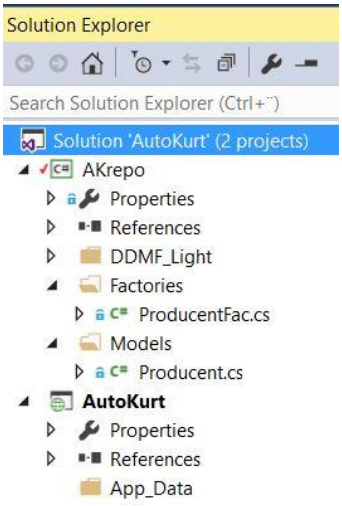
1) Almindelig klasse

```
1  namespace AKrepo.Factories
2  {
3      class ProducentFac
4      {
5          //Indsæt funktioner her
6      }
7  }
```

2) Klasse der bruger AutoFac

```
1  namespace AKrepo
2  {
3      public class ProducentFac: AutoFac<Producent>
4      {
5          //Indsæt funktioner her
6      }
7  }
```

Som du kan se på billede to har vi i linjen hvor vi har skrevet vores public class skrevet at den skal nedarve fra vores AutoFac. Som du nok også har lagt mærke til har vi bedt vores AutoFac<Producent> om at bruge modellen Producent når den henter og gemmer data i databasen.



1) Tilføj en ny klasse til mappen Factories, kald den for ProducentFac.

2) Ret koden til så den ser ud som på billede 2.

Nu er vi stort set klar til at trække data ud af vores database. Klassen ProducentFac har nu en lang række funktioner som den arver fra AutoFac, her under er en liste over hvad ProducentFac kan lave med en Producent uden at tilføje yderlige kode. Vi ser senere på brug af alle auto funktioner.

Get:	Bruges til at hente en bestemt række ud fra dens ID.
GetAll:	Kan hente alle rækker i en tabel.
GetBy:	Kan hente alle rækker med en bestemt værdi i et navngivet felt.
Insert:	Kan oprette en ny række i en tabel ud fra dens model.
Update:	Kan opdatere en række i en tabel ud fra dens model.
Delete:	Kan slette en række ud fra dens ID.
CreateConnection():	Åbner en forbindelse til databasen.
Map():	Mapper data i en model. (mere om mapping senere)
MapListe():	Mapper en liste af data ud fra en model.

Her efter kommer der en række eksempler hvordan vi bruger autofunktionerne via ProducentFac. Men inden vi går i gang skal vi lige have tilføjet et layout til dit website.

1) Opret en ny mappe i Views, kald den ”Shared”.

2) Tilføj en ny Razor layout page til mappen, kald den _MainLayout.cshtml.

Vi skal ikke lave mere ved vores layout lige nu det gør vi lidt senere, det er blot for at vi ikke skal tilføje layoutet til alle vores Views senere.

Eksemplerne kommer primært til at køre i to Controllere, CarController og ProducentController. Lad os starte med at tilføje de to Controllere.

1) Opret en ny Controller med navnet CarController.

2) Opret en ny Controller med navnet ProducentController.

Brug af autofunktionen Get()

Funktionen Get() bruges til at trække en enkelt række ud fra en tabel, ud fra dens ID.

I eksemplet bruger vi ProducentFac og Producent som vi lavede i websitet AutoKurt.

1) Åben din Controller med navnet ProducentController.

2) Ret din ProducentController til så den kommer til at se ud som på billedet her under.

```
1  using System.Web.Mvc;
2  using AKrepo;
3
4  namespace AutoKurt_MVC.Controllers
5  {
6      public class ProducentController : Controller
7      {
8          ProducentFac pf = new ProducentFac();
9
10         public ActionResult VisProducent()
11         {
12             return View(pf.Get(1));
13         }
14
15     }
16 }
```

2: I denne linje tilføjer vi vores repository til siden så vi kan bruge de funktioner der er der i.

8: Her laver vi en instans af vores ProducentFac som vi navngiver pf.

10: Vi omdøber Index actionen til VisProducent

12: Her kalder vi Get() metoden fra vores ProducentFac som returnere Producenten der har ID 1, til vores view.

3) Tilføj et empty View, kald det VisProducent og tilføj vores layout der til.

4) Tilføj følgende kode til VisProducent.cshtml:

```
1  @model AKrepo.Producent
2
3  @{
4      ViewBag.Title = "VisProducent";
5      Layout = "~/Views/Shared/_MainLayout.cshtml";
6  }
7
8      <h1>@Model.Navn</h1>
9      <img src="~/Content/Billeder/Producenter/@Model.Logo" alt="Logo"/>
10
```

Prøv at køre siden og se resultatet. Du skulle gerne få udskrevet Navn og Logo fra Producenten Opel som har ID 1 i vores tabel.

Lad os lige prøve at se lidt på koden.

1: Her fortæller vi at vores data model som vi får fra vores ProducentController, er af typen/modellen Producent.

8: Vi skriver producentens navn ud i en H1.

9: Vi skriver producentens Logo ud i et img-tag.

NB: Læg godt mærke til at når vi tilføjer en model til vores View, skrives model med lille m og når vi så bruger modellen skrives det med stort M. se linje 1 og 8

Brug af autofunktionen GetAll()

Funktionen GetAll() bruges til at hente alle rækker fra en tabel. I vores tilfælde er det alle producenterne fra tabellen Producent vi vil trække ud i en lykke. Vi bruger ProducentFac og Producent modellen som vi lavede i websitet AutoKurt.

1) Tilføj en ny Action til din ProducentController som vist her under.

```
19          public ActionResult VisProducenter()
20          {
21              return View(pf.GetAll());
22          }
```

21: Her kalder vi GetAll() der returnere en liste mad alle vores producenter, vi sender dem videre til modellen i vores View som vi laver her under.

2) Tilføj et View med navnet VisProducenter, brug _MainLayout som vi lavede tidligere.

3) Tilføj følgende kode til VisProducenter.

```
1   @using AKrepo
2   @model IEnumerable<AKrepo.Producent>
3   @{
4       ViewBag.Title = "VisProducenter";
5       Layout = "~/Views/Shared/_MainLayout.cshtml";
6   }
7
8   <h2>Vis Producenter</h2>
9   <ul>
10      @{
11          foreach (Producent p in Model)
12          {
13              <li>@p.Navn</li>
14          }
15      }
16  </ul>
```

Hvis du køre siden skulle du gerne få udskrevet Navnene på alle Producenten efterfulgt i en liste.

Lad os lige prøve at på koden igennem linje for linje.

1: I denne linje tilføjer vi en reference til vores repository så Viewet kan genkende vores modeller mm. Det gør også at vi ikke behøver at skrive hele stien til vores producent model. Slet AKrepo. I linje 2.

11: Laver vi en lykke der kører alle vores producenter igennem som vi får fra vores model.

13: Her skriver vi producentens navn ud i en liste.

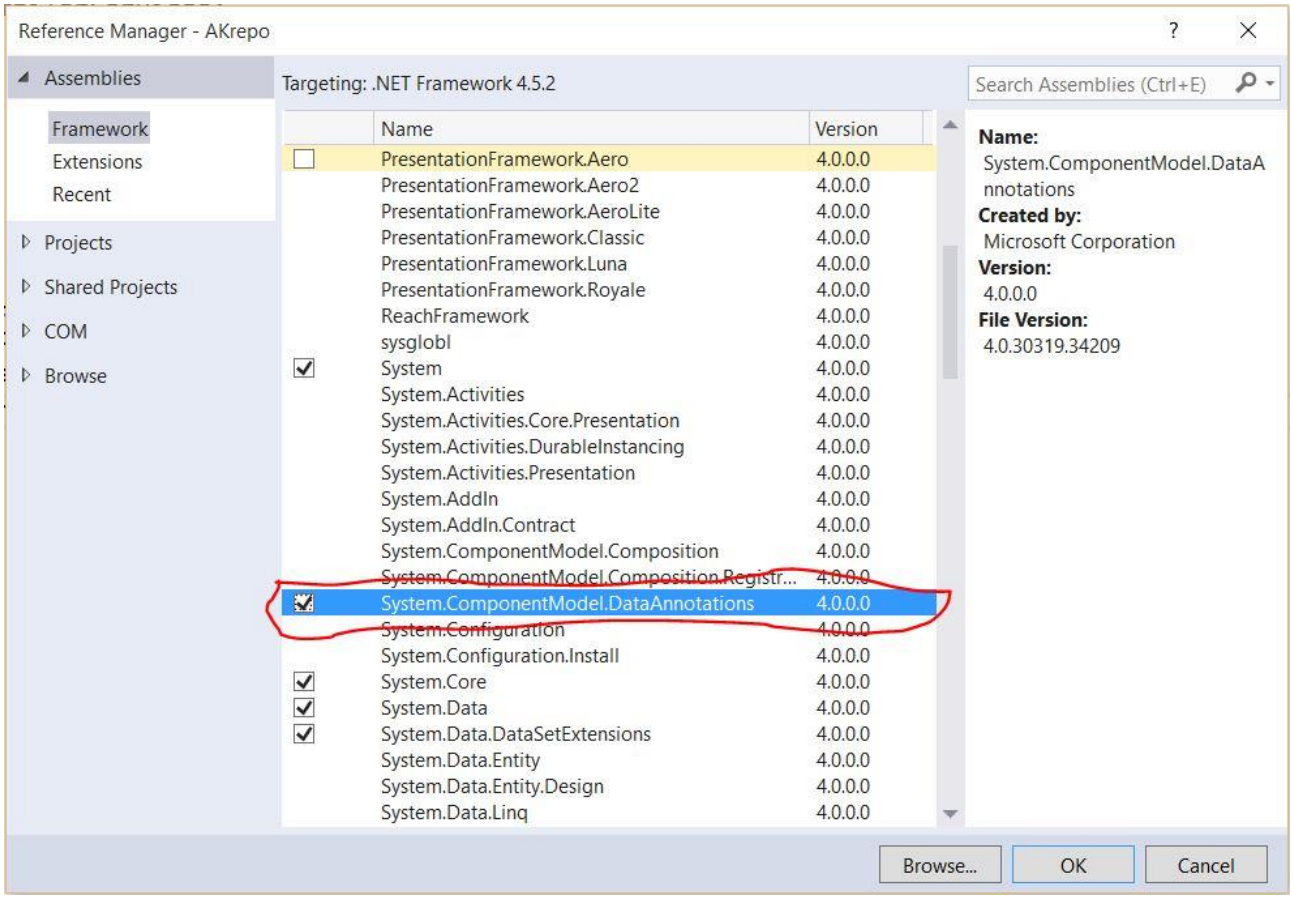
Bil modellen og BilFac

Inden vi kan begynde at arbejde med bil-tabellen, skal vi ligesom med Producenterne have oprettet en model og en factory til dette. Koden til de to klasser kommer her under.

Inden vi går videre skal vi lige have tilføjet en referance DataAnnotations i vores repository så vi kan bruge dem ligesom i vores MVC-website.

1) Højreklik på AKrepo i din Soulution Explore og vælg Add >> Reference

2) Tilføj System.ComponentModel.DataAnnotations som vist på billedet her under.



Så er vi klar til at lave vores nye Bil model.

3) Opret en klasse i mappen Models, kald den for Bil.

4) Tilføj følgende kode:

```
using System.ComponentModel.DataAnnotations;

namespace AKrepo
{
    public class Bil
    {
        public int ID { get; set; }
        [Required]
        public int ProducentID { get; set; }
        [Required]
        public string Model { get; set; }
        [Required]
        public string Beskrivelse { get; set; }
        [Required]
        public decimal Pris { get; set; }
    }
}
```

Læg mærke til vores DataAnnotation Attributs [Required]. Senere vil vi bruge den til at validere at alle properties med denne attribut, er udfyldt inden vi gemmer noget i databasen

5) Opret en klasse i mappen Factories, kald den for BilFac.

6) Ret koden til så den ser ud som følgende:

```
namespace AKrepo
{
    public class BilFac:AutoFac<Bil>
    {

    }
}
```

Kør et build på dit projekt. Nu skulle alle autofunktionerne også gerne virke på biltabellen.

Brug af autofunktionen GetBy()

Inden du laver GetBy() skal vi lige have lavet afsnittet Bil model og BilFac. Vi bruger Bil tabellen til dette eksempel da det ikke vil give mening at bruge GetBy() på Producent tabellen.

Da vi i eksemplet her under skal hente data fra to tabeller og vores View kun kan tage en Model, bliver vi nød til at lave en ViewModel. En ViewModel er en model der er samlet af andre modeller eller properties for at få det rigtige data sendt videre til vores View.

1) Tilføj en ny class til din Models-folder. Kand den for ProducentMedBiler.cs

2) Tilføj koden som der er vist her under.

```
1  using System.Collections.Generic;
2
3  namespace AKrepo
4  {
5      public class ProducentMedBil
6      {
7          public Producent Producent { get; set; }
8          public List<Bil> Biler { get; set; }
9      }
10 }
```

I eksemplet her over laver vi bare to properties der indeholder vores Producent og en liste af vores Bil-model så vi kan få begge dele i vores View.

Det er nu at GetBy() kommer ind i billedet. Vi skal nemlig have fyldt det rigtige data i vores ViewModel og der skal vi blandt andet bruge GetBy() til at hente alle de biler der har en bestemt producent. For ikke at få for meget kode i vores controller laver vi en metode i ProducetFac som samler Producent og Bil data i vores ViewModel.

1) Åben din ProducentFac.cs.

2) Lav en instans af din BilFac i ProducentFac.

3) Tilføj følgende kode:

```
1  namespace AKrepo
2  {
3      public class ProducentFac: AutoFac<Producent>
4      {
5          BilFac bf = new BilFac();
6
7          public ProducentMedBil HentMedBil(int producent)
8          {
9              ProducentMedBil pmb = new ProducentMedBil();
10             pmb.Producent = Get(producent);
11             pmb.Biler = bf.GetBy("ProducentID", producent);
12
13             return pmb;
14         }
15
16     }
17 }
```

9: Vi laver en instans af vores ViewModel. Vi kalder den pmb.

10: Her bruger vi Get() til at udfylde Producent propertyen i vores ViewModel

11: Her kalder vi GetBy() der laver en liste af alle de biler der har ProducentID 2 og putter den i vores ViewModel.

4) Tilføj en ny Action som vist her under til din ProducentController.

```
27         public ActionResult VisBilEfterProducent()
28         {
29             return View(pf.HentMedBil(2));
30         }
```

29: Her kalder vi den metode vi lige har lavet i vores ProducentFac. Metoden returnere producenten med id 2 til os som vi sender videre til vores View.

5) Tilføj et nyt View med navnet VisBilEfterProducent.cshtml

6) Indtast koden fra billedet her under.

```
1  @using AKrepo
2  @model ProducentMedBil
3  @{
4      ViewBag.Title = "VisBilEfterProducent";
5      Layout = "~/Views/Shared/_MainLayout.cshtml";
6  }
7
8  <h2>Vis Bil Efter Producent</h2>
9  <ul>
10     @{
11         foreach (Bil bil in Model.Biler)
12         {
13             <li>@Model.Producent.Navn @bil.Model</li>
14         }
15     }
16 </ul>
```

Hvis du køre siden skulle du gerne få udskrevet Navnene på alle Producenten i en liste.

Lad os lige prøve at se koden igennem linje for linje.

2: I denne linje fortæller vi at vores View skal bruge den ViewModel vi lige har lavet.

11: Laver vi en lykke der kører alle vores biler igennem som har ProducentID 2.

13: Her skriver vi Producentens navn og Bilens model ud i listen.

Brug af autofunktionen Insert()

Med autofunktionen Insert() er det muligt at oprette en ny række i en tabel. Lad os prøve at se på et eksempel hvor vi bruger Indsert() til at sætte en ny producent ind i producent-tabellen.

1) Opret en ny action i din ProducentController. Kald den for NyProducent som vist her under.

```
36          public ActionResult NyProducent()
37          {
38              return View();
39          }
```

2) Tilføj følgende et nyt View, kald det for NyProducent.

3) Tilføj koden her under til dit nye View.

```
1   @{
2       ViewBag.Title = "NyProducent";
3       Layout = "~/Views/Shared/_MainLayout.cshtml";
4   }
5
6   <h2>Ny Producent</h2>
7
8   <form name="myForm" action="/Producent/NyProducent/" method="post">
9       Navn: <input type="text" name="navn" rel="tekst"/>
10      <br/>
11      Logo: <input type="text" name="logo"/>
12      <br/>
13      <input type="submit" value="Gem"/>
14      @ViewBag.MSG
15  </form>
```

4) Tilføj en ny Action til din ProducentController som vist her under.

```
38          [HttpPost]
39          public ActionResult NyProducent(Producent p)
40          {
41              pf.Insert(p);
42
43              ViewBag.MSG = "Producenten er nu oprettet!!!";
44              return View();
45          }
```

Prøv at køre side og se om du kan oprette en ny producent.

39: I denne linje tilføjer vi en ny action der modtager en producent fra vores form.

41: Her indsætter vi den nye producent i databasen.

43: Vi udskriver en tekst i vores ViewBag for at gøre opmærksom på at producenten er oprettet.

Læg mærke til at fordi at vores View er typestærkt (@model AKrepo.Producent) så finder vores Controller selv ud af at samle det data der bliver sendt fra vores form, til en Producent i variablen p som vi kan smide direkte i metoden Insert(p).

En anden smart ting er at vi har to metoder der hedder det samme. Den eneste grund til at vi kan have det er fordi de har forskellige parametre. Læg også mærke til at den ene metode har en HttpPost attribut på. Det gør at det er denne metode der vil blive kørt når der bliver postet data og den anden når vi ikke har postet data.

Brug af autofunktionen Update()

Med autofunktionen Update() kan vi opdatere række i en tabel. I eksemplet her under prøver vi at opdatere den producent som har ID 2. Du må lige tage højde for om der stadig er en producent med ID 2 i tabellen, så tjek det lige i databasen og ret tallet til så du opdatere en række som findes.

1) Tilføj en ny action til din ProducentController. Kald den for UpdateProducent

2) Brug Get-metoden til at sende Producenten med ID 2 til et nyt View som vist her under.

```
48          public ActionResult UpdateProducent()
49          {
50              return View(pf.Get(2));
51          }
```

3) Tilføj et nyt View, kald det for UpdateProducent.

4) Tilføj følgende kode:

```
1   @model AKrepo.Producent
2   @{
3       ViewBag.Title = "UpdateProducent";
4       Layout = "~/Views/Shared/_MainLayout.cshtml";
5   }
6
7   <h2>UpdateProducent</h2>
8
9   <form name="myForm" action="/Producent/UpdateProducent/" method="post">
10      Navn: <input type="text" name="navn" value="@Model.Navn"/>
11      <br/>
12      Logo: <input type="text" name="logo" value="@Model.Logo"/>
13      <br/>
14      <input type="submit" value="Gem"/>
15      @ViewBag.MSG
16
17      <input type="hidden" id="ID" name="ID" value="@Model.ID"/>
18  </form>
```

Prøv at køre siden og se om ikke dine formularfelter er udfyldt.

Det næste vi skal er at gemme de redigerede data igen. Til det opretter vi en ny action i vores ProducentController.

1) Tilføj følgende action til din ProducentController:

```
53          [HttpPost]
54          public ActionResult UpdateProducent(Producent p)
55          {
56              pf.Update(p);
57
58              ViewBag.MSG = "Producenten er nu opdateret!!!";
59              return View(pf.Get(2));
60          }
```

Prøv at køre siden og se om du kan opdatere en producent.

56: Her updater vi den producent vi får tilbage fra vores View i variablen p.

58: Her udskriver vi en tekst i vores ViewBag.

59: Vi kalder vores View igen og fodre den med den opdaterede producent.

Brug af autofunktionen Delete()

For ligesom at få det hele med har vi også brug for at kunne slette i en tabel. Det kan vi nemt gøre med autofunktionen Delete(). Se eksemplet her under. Husk at du også her bliver nød til at tage højde for om rækken med det på gældende ID eksistere så de i databasen.

1) Tilføj en ny action til din ProducentController. Kald den for SletProducernt

2) Brug GetAll-metoden til at sende alle Producenter til et nyt View som vist her under.

```
63          public ActionResult SletProducent()
64          {
65              return View(pf.GetAll());
66          }
```

65: Vi sender en liste med alle producenter til det View som vi laver her under.

3) Tilføj et nyt View med navnet SletProducent.

4) Tilføj følgende kode:

```
1  @using AKrepo
2  @model IEnumerable<Producent>
3  @{
4      ViewBag.Title = "SletProducent";
5      Layout = "~/Views/Shared/_MainLayout.cshtml";
6  }
7
8  <h2>Slet Producent</h2>
9
10 <ul>
11     @foreach (Producent p in Model)
12     {
13         <li>@p.Navn <a href="/Producent/SletProducentSys/@p.ID/">slet</a></li>
14     }
15 </ul>
```

Lad os lige prøve at på koden igennem linje for linje.

1: I denne linje tilføjer vi en reference til vores repository så vi kan bruge de Models der er der i.

2: Vores Model skal være en liste af typen Producent.

11: Vi looper alle producenter ud og laver et link som sender producentens id til metoden SletProducentSys som vi laver nu.

1) Åben din ProducentController og tilføj en ny action som vist her under:

```
64          public ActionResult SletProducentSys(int id)
65          {
66              pf.Delete(id);
67
68              return RedirectToAction("SletProducent");
69          }
```

71: Kalder vi autofunktionen Delete() med ID'et på den række vi vil have slettet .

73: Her stiller vi tilbage til vores View der henter listen igen. Læg mærke til at vi bruger RedirectToAction som kalder en action via dens navn.

Foto Bloggen

For lige at komme til at arbejde, med de funktioner vi har set ind til nu, har jeg lavet en lille opgave her under som kommer de fleste af dem igennem.

Lav en lille blog ud fra skitsen her under. Duser indeholder alle de funktioner du skal bruge for at lave bloggen

Det skal være muligt at læse, skrive, redigere og slette i bloggen. Alle indlæg skal have en dato, overskrift og en tekst som vist på skitsen

Opret et nyt website og repository til bloggen så du får øvet dig i at starte et nyt projekt op.

Byg dine egne funktioner

Ud over alle de indbyggede funktioner, har du selvfølgelig også mulighed for at skrive dine egne. Her under laver vi to simple eksempler på hvordan du kan bygge dine egne funktioner ind i dit repository. Grunden til dette kan være hvis at AutoFac ikke kan udføre opgaven eller for at få kode flyttet fra Controlleren over i dit Repository.

Hvis jeg ikke har nævnt det før så er der et par retningslinjer man kan følge for at få en god struktur i projektet.

Viewet skal være dumt. Prøv så vidt muligt at holde dine Views typestærke og ikke for meget databehandling hvis det kan undgås.

Controlleren skal være let. Det vil sige at du skal prøve om ikke du kan lægge løsningen af forskellige opgaver ud i klasser og så kalde dem fra dine actions.

Modellerne skal være tyk. DB-logik og andet behandling af data lægges i factories og models i dit Repository.

Den første funktion vi skal se på, kan hente navnet på en kategori ud fra ID'et på denne. Det kan nogle gange være nødvendigt kun at hente et felt fra en tabel og det kan du gøre på denne måde.

1) Åben din ProducentFac.cs.

2) Du har muligvis en metode mere end vist på billedet her unde men tilføj koden fra linje 7-28 fra billedet her under:

```
1  using System.Data.SqlClient;
2
3  namespace AKrepo
4  {
5      public class ProducentFac:AutoFac<Producent>
6      {
7          public string GetName(int ID)
8          {
9
10             using (var cmd = new SqlCommand("SELECT Navn FROM Producent WHERE ID=@ID", Conn.CreateConnection()))
11             {
12                 cmd.Parameters.AddWithValue("@ID", ID);
13
14                 var r = cmd.ExecuteReader();
15                 string navn = "";
16
17                 if (r.Read())
18                 {
19                     navn = r["Navn"].ToString();
20                 }
21
22                 r.Close();
23                 cmd.Connection.Close();
24                 return navn;
25
26             }
27
28         }
29     }
30 }
```

1: Her sørger vi lige for at have adgang til SqlClient hvor vores SqlCommand ligger i, den bruger vi om lidt.

7: i denne linje laver vi en ny metode som vi kalderGetName() den modtager et tal i variablen ID og skal returnere en string.

10: Her laver vi en ny SqlCommand (cmd) i en ”Using”. Vi skriver vores Sql-forespørgsel i cmd og kalder vores CreateConnection() som laver en forbindelse til databasen.

12: Her tilføjer værdien fra variablen ID, til parametret @ID.

14: Laver vi en ny Reader som vi kalder r. I den eksekvere vi vores cmd der giver adgang til vores data i databasen via vores SQL og connection.

15: Her laver vi en tom variabel som vi skal bruge til navnet hvis der bliver fundet et.

17: Laver vi en betingelse der bliver opfyldt hvis der er fundet noget i databasen.

19: Her skriver vi navnet fra tabellen over i variablen Navn.

22: Her lukker vi vores Reader igen.

23: Lukker vores database-forbindelse efter os.

24: Til slut returnere vi variablen ”navn”.

Så mangler vi bare en Action og et View så vi kan se om vores funktion virker.

1) Lav en ny Action i din ProducentController. Kald den for HentNavn. Og kald vores nye metode i den som vist her under.

```
76          public ActionResult HentNavn()
77          {
78              ViewBag.Prod = "<b>" + pf.GetName(2) + "</b>";
79              return View();
80          }
81      }
```

Hvis vi bare har en enkelt linje der skal skrives ud behøver vi ikke lave en model og skyde i Viewet. Her bruger vi bare en Viewbag.

2) Tilføj et View til din Action med samme navn.

3) Tilføj følgende kode til dit View.

```
1  @{
2      ViewBag.Title = "HentNavn";
3      Layout = "~/Views/Shared/_MainLayout.cshtml";
4  }
5
6  <h2>HentNavn</h2>
7
8  @Html.Raw(ViewBag.Prod)
```

8: Her laver vi en ViewBag der gerne må udskrive HTML.

Eksemplet her over hentede vi kun en række fra en tabel. Lad os prøve at lave en funktion der kan hente flere rækker.

I eksemplet her under laver vi en simpel søgefunktion der kan søge på beskrivelserne i Bil tabellen.

1) Åben din BilFac.cs.

2) Tilføj følgende kode:

```
1  using System.Collections.Generic;
2  using System.Data.SqlClient;
3
4  namespace AKrepo
5  {
6      public class BilFac : AutoFac<Bil>
7      {
8          public List<Bil> Search(string keyWord)
9          {
10             using (var cmd = new SqlCommand("SELECT * FROM Bil WHERE Beskrivelse Like @keyword", Conn.CreateConnection()))
11             {
12                 cmd.Parameters.AddWithValue("@keyword", "%" + keyWord + "%");
13                 var mapper = new Mapper<Bil>();
14                 List<Bil> list = mapper.MapList(cmd.ExecuteReader());
15                 cmd.Connection.Close();
16                 return list;
17             }
18         }
19     }
20 }
```

1: Vi sikre os adgang til List<> som ligger i namespacet System.Collection.Generic.

2: Her sørge vi lige for at have adgang til SqlClient.

8: i denne linje laver vi en ny metode som vi Search(), den modtager en variablen med et søgeord og skal returnere en liste af modellen Bil.

10: Her laver vi en ny SqlCommand (cmd) i en Using. Vi skriver vores Sql-forespørgsel i cmd og kalder vores CreateConnection() som laver en forbindelse til databasen.

12: Her tilføjer værdien fra variablen keyWord, til parameteret @keyword.

13: Instanser vi vores Mapper og fortæller den at vi vil have at den skal bruge modellen Bil når den skal mappe data i vores liste.

14: Her laver vi en ny Liste af modellen Bil. Vi kalder vores MapListe() i mapperen og fodre den med resultatet fra vores cmd.

15: Lukker vores database-forbindelse efter os.

24: Til slut returnere, vi vores liste med biler.

For at teste vores lille søgefunktion bliver vi lige nød til at have sat en Action og et View op.

1) Opret en ny action med navnet Soeg i din CarController.

2) Tilføj et View med navnet Soeg.

3) Tilføj en ny action med navnet SoegResult til din CarController og tilføj følgende kode.

```
88          BilFac bf = new BilFac();
89          [HttpPost]
90          public ActionResult SoegResult(string keyWord)
91          {
92              return View("Soeg", bf.Search(keyWord));
93          }
```

90: Husk at variablen keyWord skal skrives nøjagtig som name på dit søgefelt.

4) Tilføj nu følgende kode til dit Soeg View.

```
1   @using AKrepo
2   @model List<Bil>
3   @{
4       ViewBag.Title = "Soeg";
5       Layout = "~/Views/Shared/_MainLayout.cshtml";
6   }
7   <h2>Søg</h2>
8
9   <form name="myForm" action="/Car/SoegResult/" method="post">
10      Navn: <input type="text" name="keyWord" />
11      <input type="submit" value="Søg" />
12  </form>
13
14  @{  if (Model != null)
15      {
16          if (Model.Count > 0)
17          {
18              <ul>
19                  @{
20                      foreach (Bil b in Model)
21                      {
22                          <li>@b.Model</li>
23                      }
24                  }
25              </ul>        }
26          else
27          {
28              <p>Der blev ikke fundet noget på din søgning!</p>
29          }
30      }
31  }
```

Lad os lige prøve at på koden igennem linje for linje.

14: Når siden køres første gang er der jo ikke udført en søgning så der vil ikke være nogen model på siden og der med ingen data at loope ud så det bliver vi nød til at lave et tjek på.

16: Hvis der eksistere en model så har vi lavet en søgning og her finder vi så ud af om der blev fundet noget på den ved at se om vores Search() returnere nogle rækker.

20: I denne linje looper vi de rækker der blev fundet igennem.

22: Her skriver vi model-navnet ud på de fundene biler.

Mapping

Du har nok ikke kunne undgå at høre at jeg jævnligt bruger ordet mapping så her kommer en lille forklaring.

Mapping kan ske på mange forskellige måder men i store træk går det ud på at lave et system så dit repository kan finde ud af hvad det skal gøre ved data fra databasen. I de fleste til fælde bruger vi mapping når vi skal have data fra en tabel og over i en Property-klasse også kaldt en Model. Her er det vigtigt at vores repository ved at feltet Navn i en tabel skal i Propertien Navn i en Model. Dette kan gøres på forskellige måder, nogen laver et XML-ark der bestemmer hvilke properties og felter der hører sammen fra hvilke tabeller.

Hvis vi tager udgangspunkt i vores Mapper.cs, så er den lavet på den måde at Navnet på en Model er den tabel den skal finde data i og navnene på de forskellige felter er de samme som vores property-navne så man kan sige at vores Model er et map til hvordan data skal mappes.

```
1  namespace AKrepo
2  {
3      public class Producent
4      {
5          public int ID { get; set; }
6          public string Navn { get; set; }
7          public string Logo { get; set; }
8      }
9  }
```

Producent

Navn:	Type:
ID	Int
Navn	VarChar
Logo	VarChar

Billedet her viser sammenhængen mellem vores Model og tabellen i databasen.

Manuel mapping

Ind imellem kan man have brug for at mappe noget manuelt. Lad os prøve at se på et lille eksempel hvor vi manuelt prøver at lave det mapping arbejde som vores mapper plejer at lave for os. For at det ikke skal blive for stort og tungt prøver vi at lave en metode der svare til GetAll() autofunktionen på en Producent.

1) Åben din ProducentFac.cs.

2) Tilføj følgende kode efter linje 30.

```
30
31          public List<Producent> GetAllProducent()
32          {
33              string Sql = "SELECT * FROM Producent";
34
35              SqlCommand cmd = new SqlCommand(Sql, Conn.CreateConnection());
36
37              var r = cmd.ExecuteReader();
38              var list = new List<Producent>();
39
40              while (r.Read())
41              {
42                  Producent producent = new Producent();
43                  producent.ID = int.Parse(r["ID"].ToString());
44                  producent.Navn = r["Navn"].ToString();
45                  producent.Logo = r["Logo"].ToString();
46                  list.Add(producent);
47              }
48
49              return list;
50          }
51
```

31: I denne linje laver vi en ly metode der returnere en liste af typen Producent.

33: Her laver vi en variabel med en sql-sætning der henter alt fra tabellen Producent..

35: I denne linje laver vi en ny SqlCommand og fodre den med vores sql og database-forbindelse.

37: Her laver vi en ny reader og eksekvere vores cmd der i.

38: her laver vi en ny liste af typen Producent.

40: Her laver vi en lykke der looper alle de producenter der er i vores reader r, igennem.

42: Inde i lykken laver vi en ny producent.

43-45: Mapper vi data fra readeren r over i vores producent.

46: Her smider vi producenten i listen og lykken starter forfra igen.

49: Her returnere vi listen med producenter.

Nyttige funktioner

Her under har jeg samlet en række nyttige funktioner som det kunne være en god ide at du skrev ind, for at få lidt rutine med at skrive dine egne funktioner.

Random

Den første funktion vi skal se på er en random-funktion, den returnere x antal tilfældige biler fra bil-tabellen.

1) Åben din BilFac.cs.

2) Tilføj følgende kode:

```
20          public List<Bil> GetRnd(int antal)
21          {
22              using (var cmd = new SqlCommand("SELECT TOP " + antal + " * FROM Bil ORDER BY NEWID()",
    Conn.CreateConnection()))
23              {
24
25                  var mapper = new Mapper<Bil>();
26                  List<Bil> list = mapper.MapList(cmd.ExecuteReader());
27                  cmd.Connection.Close();
28                  return list;
29              }
30
31          }
32
```

20: i denne linje laver vi en ny metode som vi GetRnd(), den modtager en variablen med et tal der skal bruges til at bestemme hvor mange rækker vi vil have returneret. Metoden returnere en liste af typen Bil.

22: Her laver vi en ny SqlCommand (cmd) i en Using. Vi skriver vores Sql-forespørgsel i cmd og kalder vores CreateConnection() som laver en forbindelse til databasen.

25: Instanser vi vores Mapper og fortæller den at vi vil have at den skal bruge typen Bil når den skal mappe data i vores liste.

26: Her laver vi en ny Liste af typen Bil. Vi kalder vores MapListe() i mapperen og fodre den med resultatet fra vores cmd.

27: Lukker vores database-forbindelse efter os.

28: Til slut returnere, vi vores liste med biler.

Count

Den næste function vi skal se på kan tælle hvor mange biler der er under en producent.

1) Åben din BilFac.cs.

2) Tilføj følgende kode:

```
34          public string GetAntal(int PID)
35          {
36              using (var cmd = new SqlCommand("SELECT Count(ID) as antal FROM Bil WHERE ProducentID=@PID",
    Conn.CreateConnection()))
37              {
38                  var r = cmd.ExecuteReader();
39                  cmd.Parameters.AddWithValue("@PID", PID);
40                  string Antal = "0";
41
42                  if (r.Read())
43                  {
44                      Antal = r["antal"].ToString();
45                  }
46
47                  r.Close();
48                  cmd.Connection.Close();
49
50                  return Antal;
51
52              }
53          }
```

34: i denne linje laver vi en ny metode som vi GetAntal(), den modtager en variablen (PID) med ID'et på den producent vi gerne vil have antallet biler på. Metoden returnere en string med antallet.

36: Her laver vi en ny SqlCommand (cmd) i en Using. Vi skriver vores Sql-forespørgsel i cmd og kalder vores CreateConnection() som laver en forbindelse til databasen.

38: Laver vi en ny reader og eksekvere vores command der i.

39: Tilføjer vi værdien fra variablen PID til parameteret @PID .

40: Vi laver en variabel til antallet.

42: Her tjekker vi om der er nogle rækker i vores reader.

44: Vi skriver antallet i variablen Antal.

47: Vi lukker vores reader igen.

48: Og lukker vores connection efter os.

50: Vi returnere indholdet af variablen Antal.

Fordi vi skriver "0" I variable Antal, vil funktionen returnere et "0" hvis den ikke finder nogle biler I database som passer sammen med den valgte producent.

Update enkelte felter

Det kan nogen gange være nødvendigt kun at opdatere nogle af de felter der er i en række. Her under ser vi på hvordan du kan opdatere en række manuelt uden brug af Autofunktionerne. I kodeeksemplet her under laver vi en funktion der opdatere Logo feltet på en producent.

1) Åben din ProducentFac.cs.

2) Tilføj følgende kode:

```
52          public void UpdateLogo(int ID, string Logo)
53          {
54              using (var cmd = new SqlCommand("UPDATE Producent SET Logo=@Logo WHERE ID=@ID",
    Conn.CreateConnection()))
55              {
56                  cmd.Parameters.AddWithValue("@Logo", Logo);
57                  cmd.Parameters.AddWithValue("@ID", ID);
58
59                  cmd.ExecuteNonQuery();
60                  cmd.Connection.Close();
61              }
62          }
```

52: i denne linje laver vi en ny metode som vi UpdateLogo(), den modtager to variabler med IDet på den producent vi gerne vil have opdateret og logoet der skal skrives i Logo.

54: Her laver vi en ny SqlCommand (cmd) i en Using. Vi skriver vores Update-sætning i cmd og kalder vores CreateConnection() som laver en forbindelse til databasen.

56-57: Tilføjer vi værdierne til de to parametre @Logo og @ID.

59: Vi eksekvere vores cmd og får skrevet logoet i databasen.

60: Og lukker vores connection efter os.

Denne funktion her vil kunne bruges sammen med en fil-upload der kan uploade logoet på en producent og derefter skrive filnavnet i databasen.

Opret og retuner ID

Ind imellem kan det være nødvendig at vide ID'et på den række der lige er blevet oprettet i databasen. Det kan vi nemt gøre ved at bruge en SQL-funktion der hedder SCOPE_IDENTITY(). Her under laver vi en lille funktion der kan oprette en producent og returnere dens ID.

1) Åben din ProducentFac.cs.

2) Tilføj følgende kode:

```
64          public string InsertAndReturnID(string Navn, string Logo)
65          {
66              string ID = "0";
67              using (var cmd = new SqlCommand("INSERT INTO Producent(Navn, Logo) VALUES(@Navn, @Logo);SELECT
    SCOPE_IDENTITY() AS curID;", Conn.CreateConnection()))
68              {
69                  cmd.Parameters.AddWithValue("@Navn", Navn);
70                  cmd.Parameters.AddWithValue("@Logo", Logo);
71
72                  var r = cmd.ExecuteReader();
73
74                  if (r.Read())
75                  {
76                      ID = r["curID"].ToString();
77                  }
78                  r.Close();
79                  cmd.Connection.Close();
80              }
81
82              return ID;
83          }
84
```

64: i denne linje laver vi en ny metode som vi kalder InsertAndReturnID(). Metoden modtager to variabler, Navn og Logo. Metoden returnere en string med antallet.

66: Her laver vi en variabel, ID som vi sætter til 0.

67: Her laver vi en ny SqlCommand (cmd) i en Using. Vi skriver vores Insert-sætning som skriver i tabellen Producent efterfulgt af en Select-sætning der finder det id der netop er oprettet. Til sidst kalder vi vores CreateConnection().

69-70: Tilføjer vi værdierne fra variablerne til parametrene.

72: Laver vi en ny reader og eksekvere vores command der i.

74: Her tjekker vi om der er nogle rækker i vores reader.

76: Vi skriver det nye ID i variablen ID.

78: Vi lukker vores reader igen.

79: Og lukker vores connection efter os.

82: Vi returnere indholdet af variablen ID.

Slet fra flere tabeller

Nogen gange kan det være nødvendigt at slette i flere tabeller. Et eksempel kan være hvis vi sletter en Producent, så vil vi også gerne have slettet de biler der ligger under producenten. I eksemplet her under bruger vi autofunktionen Delete() til at slette vores producent, efterfulgt af en ny Command der sletter bilerne fra Bil-tabellen.

1) Åben igen din ProducentFac.cs.

2) Tilføj følgende kode:

```
85          public void DeleteWithCars(int ID)
86          {
87              Delete(ID);
88
89              using (var cmd = new SqlCommand("DELETE FROM Bil WHERE ProducentID=@PID", Conn.CreateConnection()))
90              {
91                  cmd.Parameters.AddWithValue("@PID", ID);
92                  cmd.ExecuteNonQuery();
93                  cmd.Connection.Close();
94              }
95          }
```

85: i denne linje laver vi en ny metode som vi kalder DeleteWithCars(), den modtager en variabel med IDet på den producent vi gerne vil have slettet.

87: Her kalder vi autofunktionen Delete() som sletter producenten.

89: Her laver vi en ny SqlCommand (cmd) i en Using. Vi skriver vores Delete-sætning i cmd og kalder vores CreateConnection() ganske som vi plejer.

91: Tilføjer vi værdierne til parametret @ID.

92: Vi eksekvere vores cmd sletter bilerne i databasen.

93: Vi lukker vores connection efter os.

Men men men, vi mangler noget. Alle biler har tilknyttet x-antal billeder i billedtabellen. Dem burde vi også slette når nu vi sletter de biler de tilhøre, det ser vi på senere. Hvis vi har en Factory der nedarver fra vores AutoFac så er der en nemmer måde til at slette i flere tabeller. Se eksemplet her under.

```
85          public void DeleteWithCars(int ID)
86          {
87              Delete(ID);
88
89              BilFac bf = new BilFac();
90              bf.DeleteBy("ProducentID", ID);
91          }
```

89: Vi laver en instans af vores BilFac i vores ProducentFac så vi får adgang til alle auto-funktionern til bil-tabellen. Eksempelvis DeleteBy, se linje 90.

Hent data fra flere tabeller

Vi har også tit brug for at samle data fra flere tabeller i samme model. Et godt eksempel kan være vores biler, de har tilknyttet en billedtabel hvor i der er en række for hvert billede der høre til en bil. Kunne det ikke være smart hvis vi kunne lave en funktion der returnerede en bil med alle dens billeder. Her under er der to eksempler på hvordan vi henholdsvis kan hente en bil med tilhørende billeder og hvordan vi henter en liste af biler med billeder.

Inden vi starter skal vi have lavet BilledFac og Billede samt en ny Bil-model der kan indeholde en liste med billeder.

Billede modellen og BilledeFac

Inden vi kan begynde at arbejde med Billede-tabellen, skal vi have oprettet en model og en factory til dette. Koden til de to klasser kommer her under.

1) Opret en klasse i mappen Models, kald den for Billede.

2) Tilføj følgende kode:

```
public class Billede
{
    public int ID { get; set; }
    public int BilID { get; set; }
    public string Filnavn { get; set; }
    public string Alt { get; set; }
}
```

3) Opret en klasse i mappen Factories, kald den for BilledeFac.

4) Ret koden til så den ser ud som følgende:

```
using AKrepo;

public class BilledeFac:AutoFac<Billede>
{


}
```

Kør et build på dit projekt. Nu skulle alle autofunktionerne også gerne virke på denne tabel.

ViewModel BilMedBilleder

Til at samle vores bil og listen af de billeder som høre der til, skal vi bruge en ny ViewModel.

1) Opret en nu model i din Model mappe, kald den for BilMedBilleder.cs

2) Tilføj følgende kode:

```
1   using System.Collections.Generic;
2
3   namespace AKrepo
4   {
5       public class BilMedBilleder
6       {
7           public Bil Bil { get; set; }
8           public List<Billede> Billeder { get; set; }
9       }
10  }
```

Som du nok kan se indeholder vores nye Model en bil og en liste af billeder. Nu skal vi bare have udfyldt den.

Udtræk af en bil med billeder

Det næste vi skal have lavet er en metode i din BilFac der kan hente en enkelt bil med billeder.

1) Åben klassen BilFac.cs.

2) Tilføj følgende kode:

```
57          public BilMedBilleder GetWithImages(int ID)
58          {
59              BilledeFac bf = new BilledeFac();
60              BilMedBilleder bilMb = new BilMedBilleder();
61              bilMb.Bil = Get(ID);
62              bilMb.Billeder = bf.GetBy("BilID", ID);
63
64              return bilMb;
65          }
```

57: Her laver vi en nu metode som vi kalder GetWithImages(). Metoden modtager et ID på den bil som vi vil hente.

59: Vi laver en instans af vores BilledFac. Den skal vi bruge til at hente vores billeder. Det kan være en god ide at flytte den her linie op i toppen af klassen så du kan genbruge den.

60: Vi laver en ny BilMedBilleder og kalder den bilMb.

61: Vi udfylder bilmodellen i bilMb med den bil vi får fra Get()-metoden.

62: Vi udfylder Billeder i bilMb med listen med billeder vi får fra GetBy()-metoden i BilledFac.

64: vi returnere den udfyldte BilMedBilleder.

Lad os prøve at bruge metoden. Til det skal vi oprette en ny Action og et nyt View.

1) Opret en ny Action med navnet EnBilMedBIlleder.

2) Tilføj en følgende kode.

```
95          public ActionResult EnBilMedBilleder()
96          {
97              return View(bf.GetWithImages(2));
98          }
```

97: Her henter vi bilen med ID 2 via vores nye metode GetWithImages() og sender den videre til vores View.

1) Tilføj et nyt View, kald det EnBilMedBilleder.

2) Tilføj følgende kode

```
1  @using AKrepo
2  @model BilMedBilleder
3  @{
4      ViewBag.Title = "EnBilMedBilleder";
5      Layout = "~/Views/Shared/_MainLayout.cshtml";
6  }
7
8
9  <h2>@Model.Bil.Model</h2>
10
11 @{
12     foreach (Billede b in Model.Billeder)
13     {
14         <img src="/Content/Billeder/Biler/300/@b.Filnavn" alt="@b.Alt"/>
15     }
16 }
```

2: I denne linje fortæller vi at vores model er af typen BilMedBilleder der jo indeholder en Bol-model og en liste med Billed-modeller.

9: Her skriver vi bilmodellen ud i en h2.

12: Her laver vi en lykke der looper alle de billeder ud vi får fra modellen Billeder der ligger i vores ViewModel.

14: i denne linje skriver vi billedet ud i et image-tag.

Udtræk af flere bil med billeder

Lad os se på hvordan vi laver og bruger en metode der kan lave en hel liste af biler med billeder.

3) Åben klassen BilFac.cs.

4) Tilføj følgende kode:

```
68          public List<BilMedBilleder> GetAllWithImages()
69          {
70              BilledeFac bf = new BilledeFac();
71              List<BilMedBilleder> listBilMb = new List<BilMedBilleder>();
72
73              foreach (var bil in GetAll())
74              {
75                  BilMedBilleder bilMb = new BilMedBilleder();
76                  bilMb.Bil = Get(bil.ID);
77                  bilMb.Billeder = bf.GetBy("BilID", bil.ID);
78                  listBilMb.Add(bilMb);
79              }
80              return listBilMb;
81          }
```

68: Her laver vi en nu metode som vi kalder GetAllWithImages().

70: Vi laver en instans af vores BilledFac. Hvis ikke du har flyttet den her linie op i toppen af klassen så du kan genbruge den.

71: Vi laver en Liste af typen BilMedBilleder og kalder den listBilMb.

73: Her laver vi en lykke der looper alle biler igennem.

75: Vi laver en ny instans af BilMed Billeder.

76: Vi udfylder bilmodellen i bilMb med den bil vi får fra Get()-metoden.

77: Vi udfylder Billeder i bilMb med listen med billeder vi får fra GetBy()-metoden i BilledFac.

78: Her tilføjer vi vores udfyldte BilMedBilleder til listen listBilMb.

64: vi returnere den udfyldte liste af BilMedBilleder.

Lad os se om det virker.

3) Opret en ny Action i din CarController, med navnet FlereBilerMedBilleder.

4) Tilføj følgende kode til din Action.

```
32          public ActionResult FlereBilerMedBilleder()
33          {
34              return View(bf.GetAllWithImages());
35          }
```

5)

6) Tilføj et nyt View, kald det for FlereBilerMedBilleder.

```
1  @using AKrepo
2  @model List<BilMedBilleder>
3  @{
4      ViewBag.Title = "FlereBilerMedBilleder";
5      Layout = "~/Views/Shared/_MainLayout.cshtml";
6  }
7
8  <h1>Alle biler med billeder</h1>
9
10 @{
11     foreach (var bmb in Model)
12     {
13         <h2>@bmb.Bil.Model</h2>
14
15         foreach (var b in bmb.Billeder)
16         {
17             <img src="/Content/Billeder/Biler/300/@b.Filnavn" alt="@b.Alt" />
18         }
19
20         <hr/>
21     }
22 }
```

11: Her laver vi en lykke der looper listen af BilerMedBilleder igennem som vi får fra vores model.

12: Her skriver vi bilmodellen ud i et h2-tag.

15: Her laver vi en lykke i lykken, der looper alle de billeder ud vi får fra modellen Billeder.

17: i denne linje skriver vi billedet ud i et image-tag.

20: Vi adskiller bilerne med et hr-tag.

Hvis vi lige går tilbage og ser på metoden GetAllWithImages() så kan vi korte den lidt af. Vi har jo lige oppe over skrevet et stykke der henter Bilen og billederne, se billedet her under. De to stykker markeret er helt ens. Nederst på siden er et smartere eksempel hvor vi genbruger den ene metode.

```
57          public BilMedBilleder GetWithImages(int ID)
58          {
59              BilledeFac bf = new BilledeFac();
60              BilMedBilleder bilMb = new BilMedBilleder();
61              bilMb.Bil = Get(ID);
62              bilMb.Billeder = bf.GetBy("BilID", ID);
63
64              return bilMb;
65          }
66
67
68          public List<BilMedBilleder> GetAllWithImages()
69          {
70              BilledeFac bf = new BilledeFac();
71              List<BilMedBilleder> listBilMb = new List<BilMedBilleder>();
72
73              foreach (var bil in GetAll())
74              {
75                  BilMedBilleder bilMb = new BilMedBilleder();
76                  bilMb.Bil = Get(bil.ID);
77                  bilMb.Billeder = bf.GetBy("BilID", bil.ID);
78                  listBilMb.Add(bilMb);
79              }
80              return listBilMb;
81          }
```

En smartere løsning:

```
57          public BilMedBilleder GetWithImages(int ID)
58          {
59              BilledeFac bf = new BilledeFac();
60              BilMedBilleder bilMb = new BilMedBilleder();
61              bilMb.Bil = Get(ID);
62              bilMb.Billeder = bf.GetBy("BilID", ID);
63
64              return bilMb;
65          }
66
67
68          public List<BilMedBilleder> GetAllWithImages()
69          {
70              BilledeFac bf = new BilledeFac();
71              List<BilMedBilleder> listBilMb = new List<BilMedBilleder>();
72
73              foreach (var bil in GetAll())
74              {
75                  listBilMb.Add(GetWithImages(bil.ID));
76              }
77              return listBilMb;
78          }
```

Her tager vi og genbruger GetWithImages() til at lave vores BilMedBilleder der skal i listen.

Inner Join

I eksemplet her under laver vi en metode der joiner to tabeller. Den nemmeste måde at joine to tabeller på - hvis ikke vi skal have alle felter med - er at lave en type der indeholder de felter der skal bruges, fra begge tabeller og så lave en metode der hente dataet og kalder mapperen. Lad os se på et eksempel.

1) Opret en ny model, kald den BilMedProducent.

2) Tilføj følgende koder til modellen.

```
1  namespace AKrepo
2  {
3      public class BilMedProducent
4      {
5          public int ID { get; set; }
6          public int ProducentID { get; set; }
7          public string ProducentNavn { get; set; }
8          public string Model { get; set; }
9          public string Beskrivelse { get; set; }
10         public decimal Pris { get; set; }
11     }
12 }
```

Som du kan se på billedet har vi tilføjet navnet fra producent-tabellen til typen. Nu skal vi bare have lavet en metode der henter fra begge tabeller.

1) Åben klassen BilFac.cs.

2) Tilføj følgende kode:

```
106         public List<BilMedProducent> GetAllBilMedProducent()
107         {
108             using (var cmd = new SqlCommand("SELECT Bil.Model, Bil.ID, Bil.ProducentID, Bil.Pris,
    Bil.Beskrivelse, Producent.Navn AS ProducentNavn FROM Bil INNER JOIN Producent ON Bil.ProducentID =
    Producent.ID", Conn.CreateConnection()))
109             {
110                 var mapper = new Mapper<BilMedProducent>();
111                 List<BilMedProducent> list = mapper.MapList(cmd.ExecuteReader());
112                 cmd.Connection.Close();
113                 return list;
114             }
115         }
```

106: Her laver vi en nu metode som vi kalder GetAllBilMedProducent(). Metoden returnere en List af typen BilMedProducent.

108: Vi laver Command med en SQL-sætning hvor vi inner joiner alle biler med deres producent.

110: Her laver vi en instans af vores mapper og fortæller at det er typen BilMedProducent vi vil have mappet.

111: Vi mapper listen med biler over i en ny liste af typen BilMedProducent.

112: Vi lukker vores databaseforbindelse efter os.

113: vi returnere den udfyldte liste.

Lad os se om det virker.

1) Opret en ny Action med navnet HentBilMedProducent.

2) Tilføj følgende kode.

```
37          public ActionResult HentBilerMedProducent()
38          {
39              return View(bf.GetAllBilMedProducent());
40          }
```

3) Tilføj et View til din nye Action og tilføj følgende kode

```
@model IEnumerable<AKrepo.BilMedProducent>

@{
    ViewBag.Title = "HentBilerMedProducent";
    Layout = "~/Views/Shared/_MainLayout.cshtml";
}

<h2>HentBilerMedProducent</h2>

<ul>
    @{
        foreach (var bil in Model)
        {
            <li>@bil.ProducentNavn @bil.Model</li>
        }
    }
</ul>
```

Læg mærke til at vi lykken udskriver producentnavnet fra producent-tabellem, sammen med model fra bil-tabellen.

En Avanceret søgefunktion

Vi så tidligere på hvordan vi kunne lave en simpel søgefunktion, som søgte på et enkelt felt i databasen. Vi skal nu se på et lidt mere avanceret eksempel hvor vi stiller lidt flere kriterier til vores søgning.

Vi starter med at lave en metode i vores BilFac.cs. Metoden skal bruges til at søge på ProducentID, Pris og Beskrivelse. i vores tabel. Efter dette tilføjer vi en Action og sætte interfacet op i et View først.

1) Åben din BilFac.cs

2) Tilføj følgende kode.

```
118         public List<Bil> AdvSearch(string producent, string maxpris, string keyword)
119         {
120             string SQL = "SELECT * FROM Bil WHERE Beskrivelse LIKE @Keyword";
121             decimal max = 0;
122
123             if (maxpris != "")
124             {
125                 max = decimal.Parse(maxpris);
126                 SQL += " AND Pris <= @pris";
127             }
128
129             if (producent != "0")
130             {
131                 SQL += " AND ProducentID=@producent";
132             }
133
134             using (var cmd = new SqlCommand(SQL, Conn.CreateConnection()))
135             {
136                 cmd.Parameters.AddWithValue("@pris", max);
137                 cmd.Parameters.AddWithValue("@keyword", "%" + keyword + "%");
138                 cmd.Parameters.AddWithValue("@producent", int.Parse(producent));
139
140                 var mapper = new Mapper<Bil>();
141                 List<Bil> list = mapper.MapList(cmd.ExecuteReader());
142                 cmd.Connection.Close();
143                 return list;
144             }
145
146         }
147
```

118: Laver vi en ny metode som vi kalder AdvSearch(). Metoden returnere en liste af biler og modtager 3 parametre, ProducentID, maxpris og søgeord.

120: Her laver vi en variabel som vi skal bruge til at opbygge vores SQL-sætning i. Vi starter med at lave en søgning på beskrivelse lige som i det forrige søgeeksempel.

121: laver vi en variabel til vores maxpris og sætter den til 0.

123: Her tjekker vi om der er indtastet en maxpris.

125: Hvis der er indtastet en maxpris konvertere vi den til en decimal så datatypen passer med det prisen er sat til i vores database.

126: Vi tilføjer også lidt til vores SQL. Pris skal være mindre end eller lig med parameteret @pris.

129: Her tjekker vi om der er valgt en producent.

131: Hvis der er valgt en producent tilføjer vi lidt mere til vores SQL. Vi siger at ProducentID skal være lig med parameteret @producent.

134: Vi laver en ny SqlCommand der bruger vores SQL-sætning samt CreateConnection() fra vores Conn.cs.

136-138: Vi tilføjer værdierne til vores 3 parametre.

140: Vi laver en instans af vores mapper og beder den om at mappe Bil-typen.

141: Her laver vi en ny liste af typen bil og eksekvere vores Command der i.

142: Vi lukker vores connection igen.

143: Vi returnere listen med biler, hvis der er fundet nogle ellers er listen tom.

Inden vi går videre skal vi lige have lavet en ViewModel til det advarnces søge view vi skal lave senere. Viewet skal nemlig bruge to modeller nemlig en liste med producenter og en med de biler der bliver fundet på en søgning.

1) Opret en ny model, kald den SoegViewModel.

2) Tilføj følgende kode.

```
1   using System.Collections.Generic;
2
3   namespace AKrepo.Models
4   {
5       public class SoegViewModel
6       {
7
8           public List<Producent> Producenter { get; set; }
9           public List<Bil> Biler { get; set; }
10
11      }
12  }
```

1) Opret to ny Action i din CarController. Som vist her under.

```
42          ProducentFac pf = new ProducentFac();
43          SoegViewModel svm = new SoegViewModel();
44          public ActionResult AdvSoeg()
45          {
46              svm.Biler = null;
47              svm.Producenter = pf.GetAll();
48              return View(svm);
49          }
50
51          [HttpPost]
52          public ActionResult AdvSoeg(string maxpris, string producent, string keyword)
53          {
54              svm.Biler = bf.AdvSearch(producent, maxpris, keyword);
55              svm.Producenter = pf.GetAll();
56              return View(svm);
57          }
```

Som du kan se i Controlleren her over fylder vi kun noget i bil hvis der er lavet en søgning. Som du også kan se fylder vi producent hver gang, producenten skal bruges i vores dropdovnliste som vi laver i vores view her under.

1) Tilføj et View der til med navnet AdvSoeg.

2) Tilføj følgende kode.

```
1  @model AKrepo.SoegViewModel
2
3  @{
4      ViewBag.Title = "AdvSoeg";
5  }
6
7  <h2>Advanceret søgning</h2>
8
9  <form name="myForm" action="/Car/AdvSoeg/" enctype="multipart/form-data" method="post">
10     <select name="producent">
11         @{
12             foreach (var prod in Model.Producenter)
13             {
14                 <option value="@prod.ID">@prod.Navn</option>
15             }
16         }
17     </select>
18     <br />
19     Max pris:<br /> <input type="text" id="maxpris" name="maxpris" rel="tekst" />
20     <br />
21     Søgeord:<br /> <input type="text" id="keyword" name="keyword" />
22     <br />
23     <input type="submit" value="Søg" />
24 </form>
25
26
27 @{
28     if (Model.Biler != null)
29
30         if (Model.Biler.Count() > 0)
31         {
32             <ul>
33                 @foreach (var item in Model.Biler)
34                 {
35                     <li>@item.Model</li>
36                 }
37             </ul>
38         }
39         else
40         {
41             <p>Der blev ikke fundet noget på din søgning!</p>
42         }
43     }
44
45
46 }
```

Interfacet har en DropDown hvor vi skal have vores producenter smidt i. En TextBox til indtastning af en Maxpris og en til indtastning af et søgeord. Til sidst har vi en knap.

Fra linje 27 tjekker vi om der er lavet en søgning og om der er fundet noget, hvis der er det så skrives modelnavnene ud i en liste

Brugersystemet

En vigtig ting når du koder hjemmesider, er sikkerhed, vi vil i gennem et par eksempler her under opbygge et lille loginsystem som du kan gøre brug af til dine CMS-systemer.

Inden vi går i gang skal vi lige have lavet en Bruger-model og en BrugerFac.

Bruger og BrugerFac

1) Opret en klasse i mappen Models, kald den for Bruger.

2) Tilføj følgende kode:

```
public class Bruger
{
    public int ID { get; set; }
    public string Navn { get; set; }
    public string Email { get; set; }
    public string Adgangskode { get; set; }
    public int Type { get; set; }
}
```

3) Opret en klasse i mappen Factories, kald den for BrugerFac.

4) Ret koden til så den ser ud som følgende:

```
using AKrepo;

public class BrugerFac:AutoFac<Bruger>
{
    // Skriv din kode her!!!
}
```

Kør et build på dit projekt. Nu skulle alle autofunktionerne også gerne virke på denne tabel.

Login

I eksemplerne her under laver vi tre funktioner, en login, en side med sikkerhed og en logud-funktion men lad os starte med at se på selve loginfunktionen.

1) Åben din BrugerFac.cs

2) Tilføj følgende kode.

```
7       public Bruger LogIn(string email, string adgangskode)
8       {
9
10          using (var cmd = new SqlCommand("SELECT * FROM Bruger WHERE Email=@Email AND Adgangskode=@Adgangskode",
    Conn.CreateConnection()))
11          {
12              cmd.Parameters.AddWithValue("@Email", email);
13              cmd.Parameters.AddWithValue("@Adgangskode", adgangskode);
14
15              var mapper = new Mapper<Bruger>();
16              var r = cmd.ExecuteReader();
17              Bruger per = new Bruger();
18
19              if (r.Read())
20              {
21                  per = mapper.Map(r);
22              }
23
24              r.Close();
25              cmd.Connection.Close();
26              return per;
27
28          }
29      }
```

7: Vi laver en metode som vi kalder Login(). Den modtager email og adgangskode.

10: Her laver vi en ny Command hvor vi i SQL'en ser efter om der er en bruger som der matcher.

12-13: Tilføjer vi email og adgangskode til de to parametre vi lavede i SQL'en.

15: Vi instanser vores mapper med typen Bruger.

16: Vi eksekvere vores command i en ny reader, r.

17: Her laver vi en ny Bruger som vi kalder per.

19: Hvis der blev fundet en bruger via vores command, er der noger at læse i vores reader, det tjekker vi her.

21: Hvis der er en bruger så mapper vi ham i per.

24: lukker vores reader.

25: Lukker vores connectoin

26: Returnere per som er vores bruger med den indtastede e-mail og adgangskode.

Hvis nu ikke der bliver fundet en bruger i databasen så vil vi få retuneret en tom Bruger, det vil vi om lidt bruge til at afgøre om en bruger skal lukkes inde eller ej.

For at en bruger kan ligge ind, så skal vi først og fremmest bruge en form hvor man kan indtaste sin e-mail og adgangskode. Det laver vi her under. Der står en bruger i databasen nu med **test@test.dk** som e-mail og **admin** som adgangskode

1) Tilføj en ny Controller, kald den for BrugerController.

2) Tilføj en instans af din BrugerFac og en action til din BrugerController som vist her under.

```
14          BrugerFac bf = new BrugerFac();
15
16          public ActionResult Login()
17          {
18              return View();
19          }
```

3) Vi skal lige have lavet et par indstillinger i vores config-fil. Tilføj følgende til din Web.config. Koden skal sættes ind under <system.web>

```
<authentication mode="Forms">
      <forms loginUrl="~/Bruger/Login" timeout="2880">
      </forms>
</authentication>
```

Her over fortæller vi webstedet at hvor den skal finde vores login-side og hvilken type sikkerhed vi køre.

4) Tilføj et nyt View med navnet Login.

5) Tilføj følgende kode til dit View

```
1
2   @{
3       ViewBag.Title = "Login";
4       Layout = "~/Views/Shared/_MainLayout.cshtml";
5   }
6
7   <h2>Login</h2>
8
9   <form name="myForm" action="/Bruger/LoginResult/" enctype="multipart/form-data" method="post">
10      E-mail:<br />
11      <input type="text" id="Email" name="Email" />
12      <br /><br />
13      Password:<br />
14      <input type="password" id="Password" name="Password" />
15      <br /><br />
16      <input type="submit" value="Log-in" class="button" />
17      @ViewBag.MSG
18
19  </form>
```

Okay nu har vi en form til at indtaste e-mail og adgangskode samt en login funktion i vores factory. Nu mangler vi bare at binde det hele sammen i vores Controller.

1) Åben din BrugerController

2) Tilføj følgende kode.

```
21          [HttpPost]
22          public ActionResult LoginResult()
23          {
24              string email = Request["Email"].Trim();
25              string adgangskode = Crypto.Hash(Request["Password"].Trim());
26
27              Bruger b = bf.Login(email,adgangskode);
28
29              if (b.ID > 0)
30              {
31                  FormsAuthentication.SetAuthCookie(b.ID.ToString(), true);
32                  Session["UserID"] = b.ID;
33                  Session["UserName"] = b.Navn;
34                  Session["Type"] = b.Type;
35                  Session.Timeout = 120;
36
37                  if (!string.IsNullOrEmpty(Request.QueryString["ReturnUrl"]))
38                  {
39                      Response.Redirect(Request.QueryString["ReturnUrl"]);
40                  }
41
42                  return View("Secret");
43              }
44              else
45              {
46                  ViewBag.MSG = "Brugeren blev ikke fundet!";
47
48                  return View("Login");
49              }
50          }
```

Vi skal have fat i vores Login-funktion der for skal vi bruge en instans af vores BrugerFac.cs

24: Her fjerne vi blanke tegn før og efter det indtastede password og lægger det i en ny variabel som vi kalder email.

25: Det samme gør vi med adgangskode. Her hasher vi også adgangskoden så det er de to hash vi sammenligner og ikke det indtastede ord med hashen i databasen.

27: Vi laver en ny Bruger og kalder vores Login-funktion sammen med email og den hashet adgangskode. Funktionen retunere en udfyldt bruger hvis der er et match, ellers vil vi få en tom Bruger tilbage.

29: her tjekker vi om vores bruger ID er støre end 0. Hvis brugeren blev fundet i databasen vil hans ID stå i propertyen og den vil være støre end 0, hvis ikke er den 0.

31: Her sætter vi en AuthCookie der bruges til at styre om en bruger er logget ind eller ej.

32-34: Sætter vi tre sessions med det data vi gerne vil kunne læse om en bruger der er ligget ind. Vi bruger senere [”UserID”] til at sikre at en bruger er logget ind.

35: Her sætter vi session timeout til 120 minutter. Standart er normalt 60 hvis ikke vi gør noget.

42: vi vider stiller brugeren til en hemmelig side med sikkerhed som man kun kan komme ind på hvis man er logget rigtig ind.

46: Vi udskriver teksten ”Brugeren blev ikke fundet!” i vores literal hvis der ikke blev fundet en bruger.

Inden vi tester det hele så lad os lige lave siden med sikkerhed..

1) Tilføj en ny Action med navnet Secret lige som her under.

```
52          [Authorize]
53          public ActionResult Secret()
54          {
55              return View();
56          }
```

   Læg mærke til at vi sætter en attribut på Actionen der gør at vi kun kan få adgang hvis vi er logget ind med en AuthCookie

2) Tilføj et View med navnet Secret og eventuel en hemmelig tekst.

Du kan nu prøve at teste om din login virker med **test@test.dk** og **admin**.

Dette er en simpel form for sikkerhed, vi kommer senere til at se på andre mere avancerede måder at lave sikkerhed på.

Det sidste vi nu mangler er logud funktionen, den kommer her under.

1) Tilføj en ny Action, kald den for Logud

2) Tilføj følgende kode til din Controller.

```
58          public ActionResult Logud()
59          {
60              FormsAuthentication.SignOut();
61              Session.Contents.RemoveAll();
62              return RedirectToAction("Bruger", "Login");
63          }
```

60: Her fjerner vi AuthCookien igen og derved logger brugeren ud

61: Her sletter vi alle sessions i browservinduet.

62: Vi sender brugeren tilbage til siden med login-formen.

Opbygning af DDMF

Som udgangspunkt er det meningen at du selv skulle kunne lave og udvide de klasser der i tools-mappen. Her under vil jeg gennemgå koden i DDMF-mappen og komme med en række eksempler på hvordan du selv kan bygge nye autofunktioner.

Vi har tre filer i vores DDMF-mappe. Conn og Mapper som begge er hjælpeklasser til vores sidst klasse AutoFag.cs. AutoFag indeholder alle vores autofunktioner.

Det er en god ide hvis du selv skriver alle klasserne ind, det giver en bedre forståelse for hvad der er og sker i klasserne.

Conn.cs

Lad os starte med at se på vores Conn klasse. Denne klasse bruger vi til at styre vores forbindelse til databasen.

1) Opret et nyt Class-liberary. Kald det MyDbTools.

2) Opret en mappe med navnet DDMF i MyDbTools.

3) Opret en ny Class i mappen DDMF, kald den for Conn.cs

4) Tilføj følgende kode til Conn.

```
1   using System.Data.SqlClient;
2
3   namespace AKrepo
4   {
5       public static class Conn
6       {
7           private static SqlConnection _Con = new SqlConnection
    ("server=194.255.108.50;database=dbAutoKurt;uid=AutoKurt;pwd=eG8rYyC3;MultipleActiveResultSets=True");
8
9           public static SqlConnection CreateConnection()
10          {
11              var cn = _Con;
12              cn.Open();
13              return cn;
14          }
15      }
16  }
```

1: Her henter vi SqlClient hvor i SqlConnection ligger, den skal vi bruge nu.

7: Her laver vi en ny SqlConnection kaldet _Con som vi taster forbindelsen til vores Sql-server ind i. Det er den her linie du skal ændre hvis du skifter database.

9: Vi laver en ny metode med navnet CreateConnection() som kan retunere en SqlConnection.

11: her laver vi en ny conncetion på baggrund af den vi har i _Con.

12: Vi åbner vores connection til databasen.

13: til sidst returnere vi den åbne connection til der hvor CreateConnection() er blevet kaldt fra.

Læg mærke til at klassen, variablerne og metoden et af typen static. Det gør at vi kan nå funktionerne i klassen over alt fra vores applikation uden at instancere den, se eksemplet her under.

Ikke static metode:	Static metode:
Conn conn = new Conn(); conn.CreateConnection():	Conn.CreateConnection():

Mapper.cs

Mapperen bruges til at mappe dit indhold fra dine tabeller over i modeller og Lister. Mapperen Indeholder kun to public metoderne Map() og MapListe().

Jeg deler klassen over i flere billeder her under, så vi tager lidt af gangen. Det kan være at du har nogle småfejl i koden som først går væk når du har kodet hele filen. Lad os få skrevet mapperen ind.

1) Opret en ny klasse i mappen DDMF, kald den for Mapper.cs.

2) Tilføj følgende kode til din Mapper.cs.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Data;
4  using System.Linq;
5
6
7  namespace AKrepo
8  {
9      public class Mapper<T> where T : new()
10     {
11         private Dictionary<string, string> _mappings { get; set; }
12
13         public Mapper()
14         {
15             // Kortlæg hvilke properties der skal mappes til hvilke felter.
16             _mappings = CreateMap();
17         }
```

9: Her laver vi Klassen Mapper. Mapper modtager en model i variablen T. Modellen skal bruges til at kortlægge hvilke felter og properties der skal mappes sammen.

11: Vi laver et nyt Dictionary der skal indeholde vores map.

13: I konstroctoren kalder vi vores metode CreateMap() som returnere et map over hvordan felter og properties skal mappes. Det bliver lagt i variablen _mappings.

Map()

Den første metode i mapperen bruges til at mappe data fra felterne i tabellen over i vores model.

```
19          public T Map(IDataRecord record)
20          {
21              var item = Activator.CreateInstance<T>();
22              var itemType = item.GetType();
23
24
25              // Sæt properties på item ud fra mappings.
26              foreach (var map in _mappings)
27              {
28                  var prop = itemType.GetProperty(map.Key);
29
30                  if (record[map.Value] != DBNull.Value)
31                  {
32                      prop.SetValue(item, record[map.Value], null);
33                  }
34
35              }
36
37              return item;
38          }
39
```

19: Metoden Map() modtager en IDataRecord, det svare til en række fra en DataReader.

21: Vi instanser vores model T med navnet item så hvo i Modellen er en Vare så er item nu en vare.

22: Vi finder navnet på vores Model og putter det i variablen itenType.

26: Vi laver en lykke der køre vores map igennem.

28: Inde i lykken laver vi en ny property for hver property i vores map og kalder den prop.

30: Vi tjekker om der er noget data i det felt i tabellen som vi er nået til.

32: Her tager vi data fra feltet i tabellen og mapper det i den property det høre til i.

37: Vi returnere til sidst vores udfyldte model.

MapListe()
Den næst metode i mapperen bruges til at mappe en hel liste af modeller.

```
41          public List<T> MapList(IDataReader reader)
42          {
43              var list = new List<T>();
44
45              while (reader.Read())
46              {
47                  list.Add(Map(reader));
48              }
49
50              reader.Close();
51              return list;
52          }
53
```

41: Metoden returnere en liste at den valgte Model og modtager en IDataReader der svare til flere rækker fra en reader der læser fra en tabel.

43: Vi laver en ny liste af den valgte model T og kalder den list.

45: Vi laver en lykke der køre alle rækker igennem fra vores IDataReader

47: Her kalder vi metoden Map() som mapper data i vores modeller. Vi putter modellerne i vores liste list

50: Vi lukker vores reader efter os.

51: Til sidst returnere vi den udfyldte liste af modeller.

CreateMap()
Den sidste funktion i vores Mapper er ret vigtig. Det er den der maver det map der bestemmer hvordan vores data skal mappes sammen med vores modeller.

```
55          public Dictionary<string, string> CreateMap()
56          {
57              var mappings = new Dictionary<string, string>();
58              var props = typeof(T).GetProperties().Where(p => p.CanWrite);
59
60
61              foreach (var prop in props)
62              {
63                  mappings.Add(prop.Name, prop.Name);
64              }
65
66              return mappings;
67          }
68
69      }
70  }
```

55: Vi laver metoden CreateMap() der returnere et Dictionary. Et Dictionary kan indeholde en liste af værdier i par bestående af key og value.

57: Vi laver et nyt Dictonary og kalder det mappings.

58: V laver en instans af vores Model med alle de properties som der kan skrives i.

61: Her laver vi en lykke der køre alle properties igennem der er i vores Model.

63: Vi tilføjer føres vores propertynavn til key og vores feltnavn til value. I vores tilfelde er navnet det samm, hvis vi havde brugt prefixs i tabellerne kunne der eksempelvist se sådan her ud: mappings.Add(prop.Name, ”fld” + prop.Name);.

66: Til sidst returnere vi vores map.

AutoFac

Som jeg nok har nævnt tidligere, så er AutoFac omdrejningspunktet for hele autofunktionaliteten. AutoFac indeholder alle de metoder som bliver kaldt når du bruger en autofunktion. Mapperen og Conn klassen er kun hjælpeklasser til AutoFac.

For at lærer klassen og dens metode bedre at kende er det en god ide at taste den selv. Her under kommer kode delt op i små bider så du ikke skal forholde dig til alle 145 linjer på en gang.

1) Opret en ny klasse i mappen DDMF, kald den for AutoFac.cs.

2) Tilføj følgende kode til din AutoFac.cs.

```
1  using System.Collections.Generic;
2  using System.Data.SqlClient;
3
4  namespace AKrepo
5  {
6      public class AutoFac<T> where T : new()
7      {
8          private string table;
9          private Mapper<T> mapper = new Mapper<T>();
10
11         public AutoFac()
12         {
13             table = typeof(T).Name;
14         }
15
16
```

1: Her henter vi funktionaliteten til at arbejde med lister.

2: Vi henter også SqlClient som blandt andet indeholder SqlCommand og SqlConnection.

6: Vi laver en ny klasse med navnet AutoFag og kan modtage en Model i variablen T. Vi instanser T i samme linje så vi kan bruge den i hele klassen.

8: Her laver vi en variable der skal bruges til at opbevar navnet på den tabel der høre til Modelen T.

13: I vores Constructor sætter vi tabelnavnet lig med navnet på vores model.

Nu har vi ligesom fået styr på de informationer vi skal bruge i klassen for at lave vores autofunktioner.

Get()

Her under er den først autofunktion beskrevet, metoden kan hente en række fra en tabel. Indtast metoden Get() her under.

```
17          public T Get(int ID)
18          {
19              using (var cmd = new SqlCommand("SELECT * FROM " + table + " WHERE ID=@ID",
    Conn.CreateConnection()))
20              {
21                  cmd.Parameters.AddWithValue("@ID", ID);
22
23                  var r = cmd.ExecuteReader();
24                  T type = new T();
25
26                  if (r.Read())
27                  {
28                      type = mapper.Map(r);
29                  }
30
31                  r.Close();
32                  cmd.Connection.Close();
33                  return type;
34              }
35          }
```

17: Vi laver en ny metode som vi kalder Get(). Get retunere vores model (T) og modtager ID'et på den række den skal vinde i tabellen.

19: I vores using laver vi en ny SqlCommand, cmd med en Sql-sætning der vælger alt fra den aktuelle tabel, hvor ID er lig med værdien af parameteret @ID. Cmd får også vores databaseforbindelse via CreateConnection() i vores statiske Conn klasse.

21: Vi tilføjer værdien til parameteret @ID.

23: Vi laver en ny Reader og eksekvere vores cmd der i.

24: Vi laver en ny model.

26: Her tjekker vi om der er noget at læse i vores reader.

28: Hvis der er noget at læse kalder vi Map() i vores Mapper og får fyldt data fra databasen i vores model.

31: Vi lukker vores reader efter os.

32: Vi lukker vores connection efter os.

33: Til sidst, returnere vi vores model

Metoden Get() er bygget sådan at hvis der ikke bliver fundet noget i tabellen, så vil den returnere en tom model hvilket du så vil kunne lave et tjek på i din codebehind.

GetAll()

Den næste funktion vi skal se på er GetAll(). Den kan hente alle rækker fra en tabel og returnere dem i en Liste at den aktuelle model. Indtast koden her under.

```
37          public List<T> GetAll()
38          {
39              using (var cmd = new SqlCommand("SELECT * FROM " + table, Conn.CreateConnection()))
40              {
41                  List<T> list = mapper.MapList(cmd.ExecuteReader());
42                  cmd.Connection.Close();
43                  return list;
44              }
45          }
46
```

37: GetAll() retunere en List af modellen T.

39: I vores using laver vi en ny SqlCommand med en Sql-sætning der henter alt fra den aktuelle tabel. Cmd får igen vores databaseforbindelse via CreateConnection() i vores statiske Conn klasse.

41: Vi laver en ny List af den aktuelle model T og beder vores mapper om at mappe dataet fra vores cmd i en liste.

42: Her lukker vi vores forbindelse igen.

43: Vi returnere listen.

GetBy()
Den næste funktion fungere stort set lige som GetAll() bort set fra at den kan hente alle der har en bestemt værdi i et bestemt felt. Det kan for eksempel være hvis du vil hente alle produkter som er under en bestemt kategori.

Indtast koden fra billedet her under.

```
47
48          public List<T> GetBy(string field, int value)
49          {
50              using (var cmd = new SqlCommand("SELECT * FROM " + table + " WHERE " + field + "=@KID",
    Conn.CreateConnection()))
51              {
52                  cmd.Parameters.AddWithValue("@KID", value);
53
54                  List<T> list = mapper.MapList(cmd.ExecuteReader());
55                  cmd.Connection.Close();
56                  return list;
57              }
58          }
59
```

47: Vi laver en ny metode med navnet GetBy(), Metoden returnere en List af modellen T og modtager to variabler, et felt navn og et tal typisk et ID.

50: I vores using laver vi en ny SqlCommand med en Sql-sætning der vælger alt fra den aktuelle tabel hvor det valgte felt er lig med værdien af parameteret @KID, vi kalder også her CreateConnection() og får vores forbindelse på.

52: Vi tilføjer værdien fra value til parameteret @KID.

54: Vi laver en ny List af den aktuelle model T og beder vores mapper om at mappe dataet fra vores cmd i en liste.

55: Her lukker vi vores forbindelse igen.

56: Vi returnere listen.

Delete()
Vi har to slet funktioner i AutoFag. Den første vi skal se på er Delete(). Den kan slette en række i en tabel ud fra rækkens ID. Tilføj koden fra billedet her under.

```
60
61          public void Delete(int ID)
62          {
63              using (var cmd = new SqlCommand("DELETE FROM " + table + " WHERE ID=@ID",
    Conn.CreateConnection()))
64              {
65                  cmd.Parameters.AddWithValue("ID", ID);
66                  cmd.ExecuteNonQuery();
67                  cmd.Connection.Close();
68              }
69          }
70
```

61: Vi laver en ny metode med navnet Delete(), Metoden modtager et ID på den række vi vil have slettet.

63: I vores using laver vi en ny SqlCommand med en Sql-sætning sletter fra den aktuelle tabel, hvor ID er lig med værdien af parameteret @ID,

65: Vi tilføjer værdien fra ID til parameteret @ID.

66: Her eksekvere vi vores command som udføre sletningen i tabellen.

67: Vi lukker vi vores forbindelse igen.

DeleteBy()
Den sidste slet funktion bruges til at slette rækker ud fra et valgfrit felt der har en bestemt værdi eksempelvis hvis du vil slette alle produkter der har en bestemt producent.

```
71          public void DeleteBy(string field, int value)
72          {
73              using (var cmd = new SqlCommand("DELETE FROM " + table + " WHERE " + field + "=@value",
    Conn.CreateConnection()))
74              {
75                  cmd.Parameters.AddWithValue("@value", value);
76                  cmd.ExecuteNonQuery();
77                  cmd.Connection.Close();
78              }
79          }
```

71: Vi laver en ny metode med navnet DeleteBy(), Metoden modtager et feltnavn og en værdi på den eller de række vi vil have slettet.

73: I vores using laver vi en ny SqlCommand med en Sql-sætning sletter fra den aktuelle tabel hvor det valgte felt er lig med den den valgte værdi i parameteret @value,

75: Vi tilføjer værdien fra value til parameteret @value.

66: Her eksekvere vi vores command som udføre sletningen i tabellen.

67: Vi lukker vi vores forbindelse igen.

Insert()

Metoden Insert() bruges til at indsætte en ny række i en tabel. Metoden kræver at alle properties er udfyldt, på nær ID. Metoden modtager en Model i variablen pro og ud fra den generer den selv Sql-sætningen.

```
81          public void Insert(T pro)
82          {
83              string parms = "";
84              string fielsds = "";
85
86              var mappings = mapper.CreateMap();
87
88              foreach (var map in mappings)
89              {
90                  if (map.Key.ToLower() != "id")
91                  {
92                      fielsds += map.Value + ", ";
93                      parms += "@" + map.Key + ", ";
94                  }
95              }
96
97              fielsds = fielsds.Substring(0, fielsds.Length - 2);
98              parms = parms.Substring(0, parms.Length - 2);
99
100             using (var cmd = new SqlCommand("INSERT INTO " + table + " (" + fielsds + ") VALUES(" + parms
    + ")", Conn.CreateConnection()))
101             {
102
103                 foreach (var prop in mappings)
104                 {
105                     if (prop.Key.ToLower() != "id")
106                     {
107                         cmd.Parameters.AddWithValue(prop.Key, pro.GetType().GetProperty
    (prop.Key).GetValue(pro, null));
108                     }
109                 }
110
111                 cmd.ExecuteNonQuery();
112                 cmd.Connection.Close();
113             }
114         }
115
```

83-84: Her laver vi 2 variabler. Den ene skal bruges til at opbevare vores parametre navne og den anden til vores feltnavne som vi skal bruge for at opbygge vores SQL.

86: Vi kalder vores Mapper og via CreateMap() metoden får vi sendt et Dictionary tilbage med alle properties og felter der er i vores map.

88: Vi laver en lykke der køre vores map igennem.

90: Vi sørge for at vores ID-felt ikke bliver en del at vores SQL, vi kan jo ikke inserte i ID.

92: I variablen fields tilføjer vi alle vores feltnavne fra vores map, adskilt af et komma og mellemrum.

93: I variablen parms til føjer vi alle vores propertinavne som parametre fra vores map, adskilt af et komma og mellemrum.

97-98: Fjerner vi de sidste komma og mellemrum i de to variabler.

100: Her laver vi en ny command hvor i vi opbygger vores insert sql med tabel navnet, feltnavne og parametre fra de tre variabler. Vi kalder også CreateConnection() for at få adgang til vores database.

103: Vi skal have tilføjet værdierne til vores parametre så vi laver igen en lykke der køre vores map igennem.

105: Vi sørge for at vores ID-felt ikke kommer med da vi ikke har et ID-parameter.

107: Vi tilføjer værdierne til vores parametre inde i lykken.

111: Vi eksekvere vores command og får skrevet rækken i tabellen.

112: Her lukker vi forbindelsen efter os.

Update()
Den sidste metode i AutoFac bruges til at opdatere en række i en tabel. Metoden modtager en udfyldt model som den skriver ind i den række hvor ID-propertyen høre til. Så i modsætning til Insert() skal ID være udfyldt her.

```
116         public void Update(T pro)
117         {
118             string fAndP = "";
119             var mappings = mapper.CreateMap();
120 
121             foreach (var map in mappings)
122             {
123                 if (map.Key.ToLower() != "id")
124                 {
125                     fAndP += map.Value + "=@" + map.Key + ", ";
126                 }
127             }
128 
129             fAndP = fAndP.Substring(0, fAndP.Length - 2);
130 
131             using (var cmd = new SqlCommand("UPDATE " + table + " SET " + fAndP + " WHERE ID=@Id",
    Conn.CreateConnection()))
132             {
133 
134                 foreach (var prop in mappings)
135                 {
136                     cmd.Parameters.AddWithValue(prop.Key, pro.GetType().GetProperty(prop.Key).GetValue
    (pro, null));
137                 }
138 
139                 cmd.ExecuteNonQuery();
140                 cmd.Connection.Close();
141             }
142         }
143 
144     }
145 }
```

118: Her laver vi en variabel der skal bruges til at opbevare vores parametre og feltnavne som vi skal bruge for at opbygge vores SQL.

119: Vi kalder vores Mapper og via CreateMap() metoden får vi sendt et Dictionary tilbage med alle properties og felter der er i vores map.

121: Vi laver en lykke der køre vores map igennem.

123: Vi sørge for at vores ID-felt ikke bliver taget med.

125: I variablen fAndP tilføjer vi alle vores feltnavne og vores propertynavn adskilt af ”=@” så der eksempelvis kunne komme til at stå Navn=@Navn.

129: Fjerner vi det sidste komma og mellemrum i variablen fAndP.

131: Her laver vi en ny command hvor i vi opbygger vores update sql med, tabel navnet fra table variablen og feltnavne og parametrene fra fAndP variablen. Vi kalder også CreateConnection() for at få adgang til vores database.

134: Her skal vi have tilføjet værdierne til vores parametre, så vi laver igen en lykke der køre vores map igennem.

136: Vi tilføjer værdierne til vores parametre inde i lykken. ID skal også med denne gang så vi ved hvilken række der skal opdateres.

139: Vi eksekvere vores command og får skrevet rækken i tabellen.

140: Her lukker vi forbindelsen efter os.

Kommentering
For at få en bedre forståelse af kode er det en god ide at tage en kopi af din DDMF-mappe og efterfølgende kommentere i koden hver eneste linje.

Hvis der skulle være nogle linjer du ikke kan huske hvad gør, så slå dem op igen. Kommenter også Mapperen og Conn klassen.