

# Entity Framework



## ORM

- Object-relational mapper

## Objekter

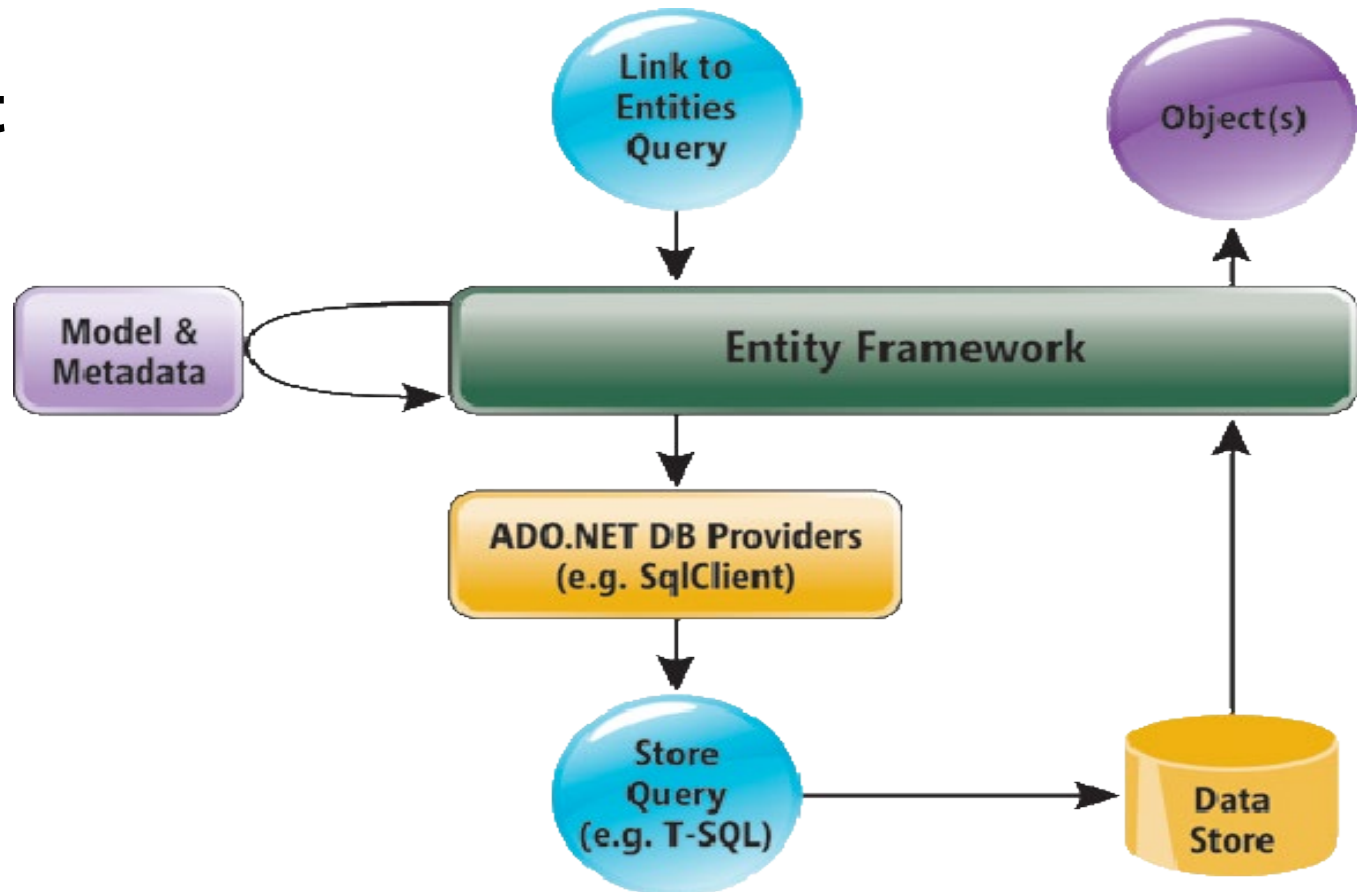
- Lever i hukommelsen
- Ikke persistente
- Stærkt typet

## Data i relational database

- Typisk gemt i filsystemet via SQL Server
- Persistent

# Hvad er Entity Framework?

## Oversigt



# Tre ORM strategier

- **Code first**
- **Database first**
- **Model first ("klassediagram" først)**
  
- **Vi skal anvende Code first**



# Entities

## Entity

- Andet ord for noget vi vil gemme persistent – typisk et objekt.

## Objekter

- Instanser af klasser

## Code First

- Vi starter med at lave vores model klasser
- Hvis en klasse har en property med det rigtige navn (Id eller <klassenavn>Id, vil den property blive brugt som primary key i databasen

# Entity Framework og Projekter

## Entity Framework

- Skal tilføjes til det projekt hvor man ønsker at bruge EF (ikke med i .Net Framework)

## NuGet Package Manager

- Højreklik på projekt, vælg Manage NuGet packages...
- Søg efter og installer seneste (6.4.4, medmindre du bruger core)

# Skridt til at lave en EF applikation

**1. Installer entity framework vha. NuGet package manager (Højreklik projekt, vælg manage NuGet packages..., søg efter entityframework)**

## **2. Lav model**

1. Primary key felter : navn ID eller  
    <klassenavn>ID
2. Foreign keys og navigation properties
3. Skal have tom ctor



# Skridt til at lave en EF applikation

```
• public class Bil
• {
•     public int BilID { get; set; }
•     public string Name { get; set; }
•     public int Weight { get; set; }
•     public Bil()
•     {
•     }
•     public Bil(string name, int weight)
•     {
•         Name = name;
•         Weight = weight;
•     }
• }
```



# Skridt til at lave en EF applikation

## 3. Opret context (normalt i DAL directory)

1. Nedarv fra DbContext
2. Opret DbSet (normalt for hver tabel)



# DbContext

- **using System.Data.Entity;**
- **Holder styr på ændringer der er sket på Entities**

```
public class BilContext : DbContext
{
    public BilContext() : base("Biler")
    {
    }
    public DbSet<Bil> Biler { get; set; }
}
```

# Skridt til at lave en EF applikation

## 4. Opret initializer (valgfri):

```
internal class BilInitializer :  
    DropCreateDatabaseIfModelChanges<BilContext>  
    {  
        protected override void Seed(BilContext context)  
        {  
            context.Biler.Add(new Bil("Ford", 1400));  
            context.SaveChanges();  
        }  
    }
```

# Skridt til at lave en EF applikation

## 5. Fortæl EF, at initializeren skal køres(app.config):

```
<entityFramework>
  <providers>
    <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
  </providers>
  <contexts>
    <context type="DAL.BilContext, MyFirstEFApp">
      <databaseInitializer type="DAL.BilInitializer,
MyFirstEFApp" />
    </context>
  </contexts>
</entityFramework>
```

## 6. Lav connection string (i app.config):

```
<connectionStrings>
```

```
<add name="Biler"
```

```
    connectionString="Data Source=EAA-SH-KLB0-  
KU\SQLEXPRESS;Initial Catalog=Biler1;Integrated  
Security=SSPI;"
```

```
    providerName="System.Data.SqlClient"/>
```

```
</connectionStrings>
```

## Skridt til at lave en EF applikation

- 7. Kør projektet og se, at tabeller bliver oprettet i databasen, med evt. data.**



# DbSet

- **En central tilgang til data**
- **Sættes op i context**



## Læs alle biler

```
BilerListe.Items.Clear();  
foreach(Bil bil in context.Biler)  
{  
    BilerListe.Items.Add(bil);  
}
```

Opretter også databasen hvis den ikke findes



## Opret ny bil/gem data

```
context.Biler.Add(new Bil("Mazda", 2000));  
context.SaveChanges();
```

- **SaveChanges gemmer ændringer til data siden sidste kald til SaveChanges**

## Mere avanceret query

```
IQueryable<Bil> result = context.Biler.Where(x => x.Name ==  
NameSearch.Text);  
BilerListe.Items.Clear();  
foreach(Bil bil in result)  
{  
    BilerListe.Items.Add(bil);  
}
```

# Opgave

- **Opgave 8.1**



Data migration

# Database Migrations



# Data Migration

## Model

- Nye klasser
- Nye Properties

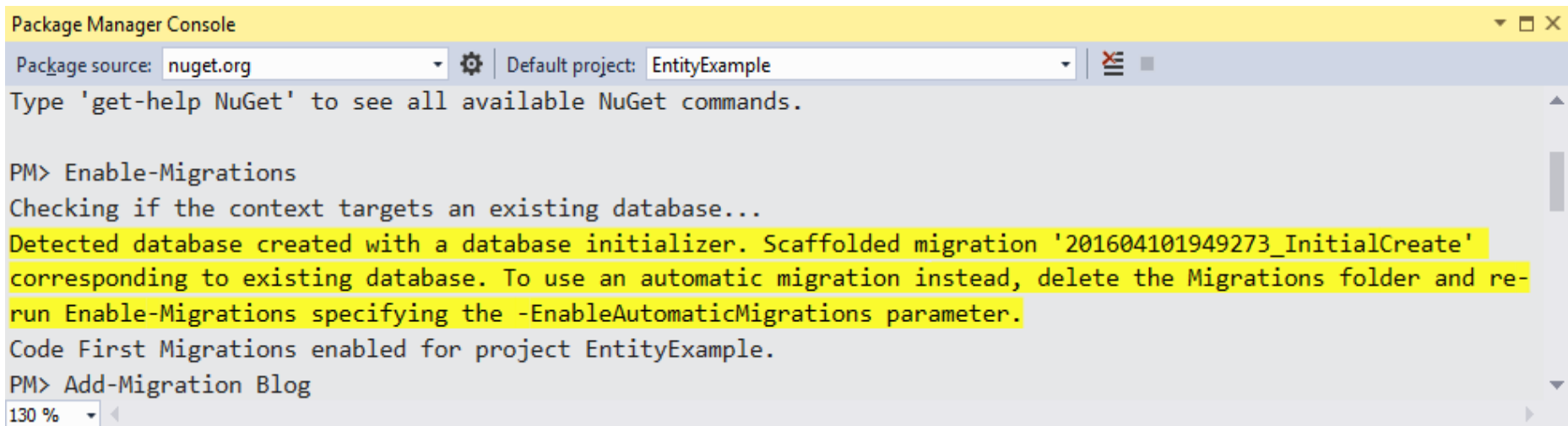
## Ændringer

- Navne
- Typer
- Annotations

# Data Migration

View -> Other Windows -> Package Manager Console

Tools -> NuGet Package Manager -> Package Manager Console



```
Package Manager Console
Package source: nuget.org | Default project: EntityExample
Type 'get-help NuGet' to see all available NuGet commands.

PM> Enable-Migrations
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration '201604101949273_InitialCreate'
corresponding to existing database. To use an automatic migration instead, delete the Migrations folder and re-
run Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project EntityExample.
PM> Add-Migration Blog
130 %
```

# Data Migration

## Enable-Migrations

- Tilføjer en Migrations folder til projektet
- Configuration.cs

## Add-Migration <migration name>)

- Parameter: Navn på migration
- Tilføjer en migration fil til Migrations folderen

## Update-Database

- Opdatere databasen med de seneste ændringer

# De 4 skridt i en migration

1. Du laver de ændringer du vil i modeller og/eller tilføjer nye modeller til context som DbSet.
2. Du laver migration filen med "add-migration <name>" command.
3. Check migration filen og se at det ser fornuftigt ud.
4. Updater databasen med migrationen i NuGet consolen med "update-database" – dette vil starte Up metoden i migrationen
5. Somme tider duer den genererede kode ikke. Jeg prøvede at ændre navnet på PK-feltet, det resulterede i, at der blev oprettet et ny felt, hvorefter det gamle blev slettet. Dvs. der var to PK samtidig. Det gik ikke, og opdateringen blev rullet tilbage. Ændring af rækkefølgen løste problemet.



# Model ændringer og migration - sammenfattet

- **Slå migration til:**
  - Vis Package manager console:  
View -> Other Windows -> Package Manager Console (eller under tools)
  - kør Enable-Migrations
  - Der dannes et directory Migration med et par filer
- **Lav ændringer til model**
- **I Package manager console, kør**
  - Add-Migration <navn på ændringer>
- **Kør Update-Database i PM console**

# Data migration

## Configuration.cs

- Seed Metode
  - Bruges til at initialisere data i databasen

## Migration klasser

- Up/Down metoder

# Eksempler på up/down i en migration

```
6 public partial class CustomerCreatedDate : DbMigration
7 {
8     public override void Up()
9     {
10         AddColumn("dbo.Customer", "CreatedDate",
11             c => c.DateTime(nullable: false, defaultValueSql: "GETUTCDATE()"));
12     }
13
14     public override void Down()
15     {
16         DropColumn("dbo.Customer", "CreatedDate");
17     }
18 }
```

**Koden er autogeneret – baseret på de ændringer du har lavet i koden i forhold til databasen. I dette tilfælde tilføjes blot et felt CreatedDate til Customer klassen.**

## Rollback

# Rollback migrations (brug af down methods)

```
PM> Get-Migrations
```

```
Retrieving migrations that have been applied to the target database.
```

```
201208012131302_Add-SystemCategory
```

```
201207311827468_CategoryIdIsLong
```

```
201207232247409_AutomaticMigration
```

```
201207211340509_AutomaticMigration
```

```
201207200025294_InitialCreate
```

```
PM> Update-Database -TargetMigration:"CategoryIdIsLong"
```

# Navigation med liste-variabel

```
BilerListe.Items.Clear();  
foreach(Bil b in ((Firma)FirmaListe.SelectedItem).Biler)  
{  
    BilerListe.Items.Add(b);  
}
```



# Mange-mange relation

- **Lav liste af den anden type i begge klasser**
  - I Firma:  
`public virtual List<Ejer> Ejere { get; } = new List<Ejer>();`
  - I Ejer:  
`public virtual List<Firma> Firmaer { get; } = new List<Firma>();`
  - Gennemløb som for en-mange relation

# Brug af data – nogle eksempler følger

Basalt set kan du bruge Alle LINQ queries på alle context tabellerne til at udvælge præcis hvad du gerne vil have.

Brug data i dit program.

---



```
PetHotelContext context = new PetHotelContext();  
  
var s = from pet in context.Pets select pet;  
  
//data kan nu bruges til f.eks. Opdatere en ListBox  
petListBox.ItemsSource = s.ToList();  
petListBox.Items.Refresh();
```

## Hente Data alt data fra tabel - LinQ eksempler

---



```
bool isChecked = (bool)IsGuestCheckBox?.IsChecked;
var s = from pet in context.Pets
        where pet.Age>=minAge && pet.Age<=maxAge &&
        pet.IsGuestNow==isChecked
        orderby pet.PetName
        select pet;

//do stuff with result
petListBox.ItemsSource = s.ToList();
petListBox.Items.Refresh();
```

## LINQ Eksempel – filtrering af data

---

# Initializers

Initializers

---

```
<entityFramework>
  <contexts>
    <context type="PetHotelWPF.DAL.PetHotelContext,
PetHotelWPF">
      <databaseInitializer
type="PetHotelWPF.DAL.PetHotelInitializer,PetHotelWPF" />
    </context>
  </contexts>
...•
```

//PetHotelWPF er navnet på solution i dette tilfælde.

## Tilføj Initializer i App.Config

---

```
public class PetHotelInitializer : CreateDatabaseIfNotExists<PetHotelContext>
{
    //other options are
    //DropCreateDataBaseAlways
    //DropCreateDataBaseIfModelChange
    {
        protected override void Seed(PetHotelContext context)
        {
            List<Customer> customers = new List<Customer>();
            Customer c1 = new Customer("martin", "aarhus");
            Customer c2 = new Customer("benny", "Esbjerg");
            customers.Add(c1);
            customers.Add(c2);

            //here we add all of our customers to the dataset in the context
            customers.ForEach(customer => context.Customers.Add(customer));

            Pet p1 = new Pet(1, "doggy", "Dog"); //belongs to customer 1
            Pet p2 = new Pet(1, "snaky", "Snake"); //belongs to customer 1
            Pet p3 = new Pet(2, "catty", "Cat"); //belongs to customer 2
            context.Pets.Add(p1);
            context.Pets.Add(p2);
            context.Pets.Add(p3);
            //saving changes
            context.SaveChanges();
        }
    } //end of class
}
```

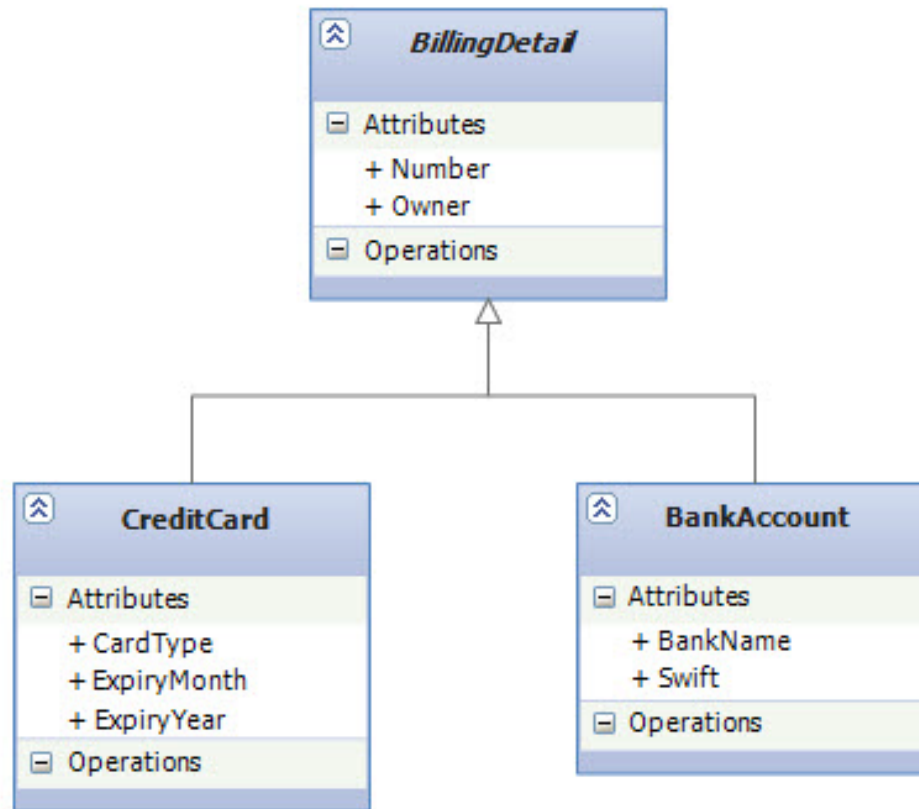
# Entity Framework og klasse hierakier (inheritance)

EF og inheritance

---



# Table per Hierarchy (TPH) – default strategi i EF.



## Inheritance og EntityFramework

---

```
public abstract class BillingDetail {
    public int BillingDetailId { get; set; }    //primary key
    public string Owner { get; set; }
    public string Number { get; set; }
}

public class BankAccount : BillingDetail {
    public string BankName { get; set; }
    public string Swift { get; set; }
}

public class CreditCard : BillingDetail {
    public int CardType { get; set; }
    public string ExpiryMonth { get; set; }
    public string ExpiryYear { get; set; } }

public class InheritanceMappingContext : DbContext {
    public DbSet<BillingDetail> BillingDetails { get; set; }
}
```

I kode vil det se sådan her ud.

---

billingDetails listen i koden nedenfor kan så indeholde både CreditCard og BankAccount objekter:

```
IQueryable<BillingDetail> linqQuery = from b in context.BillingDetails select b;  
List<BillingDetail> billingDetails = linqQuery.ToList();
```

Vi kan også vælge blot KUN at hive objekter ud af en bestemt type meget nemt:

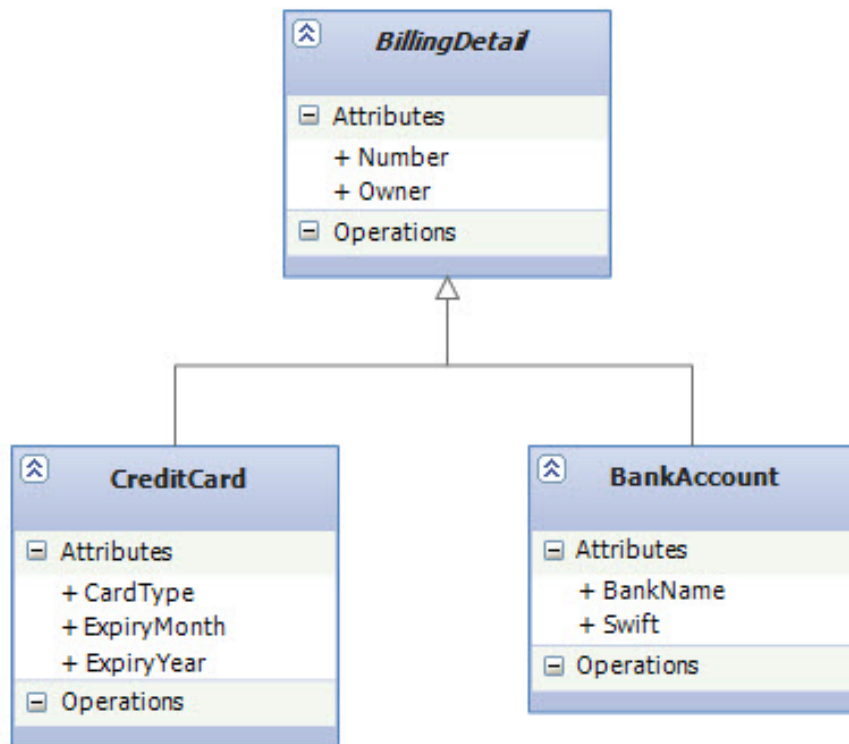
```
IQueryable<BankAccount> query =  
    from b in context.BillingDetails.OfType<BankAccount>() select b;  
List<BankAccount> accounts = query.ToList();
```

## Polymorphic queries i EF

---







BillingDetails		
	Column Name	Nullable
🔑	BillingId	No
	Discriminator	No
	Owner	Yes
	Number	Yes
	BankName	Yes
	Swift	Yes
	CardType	Yes
	ExpiryMonth	Yes
	ExpiryYear	Yes

**Discriminator felt:** styres af EF. Angiver som en string hvilket type objekt der er tale om. De andre felter er nullable, men ikke ID og discriminator

## Inheritance – hvordan ser databasen ud?

# Opgaver

Opgaver!

---

