Entity Framework

Mål med lektionen!

- Repetera och befästa kunskaperna.

Vad lektionen omfattar

- Repetera och gå igenom kursen lite snabbt.

Vilka problem vill vi **lösa**?

- **Vi arbetar med Webbapplikationer**
- Vi kommer att behöva **läsa, ändra och lägga till ny data**
  - **CRUD**
- Det **vanligaste** sättet när vi arbetar med **.NET** kommer att vara en **SQL databas**
- Vi behöver alltså ha en **kommunikation** mellan vår **webbapplikation** och **databasen**(en eller flera)

Vad är **O/RM**?

- ORM är ett verktyg för att lagra data **från domänobjekt till relationsdatabaser** som MS SQL Server, på ett automatiserat sätt utan mycket programmering.

- O/RM innehåller tre huvuddelar
  - **Domän klassobjekt**,
  - **Relationsdatabasobjekt**
  - **Mapping information**

  om hur domänobjekt mappas till relationsdatabasobjekt (tabeller, vyer och storedprocedures).

- ORM låter oss hålla vår **databas design skild från vår domän klass design**.

- Detta gör programmet **lätt att underhålla och bygga ut**.
  - Det **automatiserar också standard CRUD operationer** (Skapa, läsa, uppdatera och radera) så att *utvecklaren inte behöver skriva det manuellt.*

Vad är Entity Framework?

- Skriva och hantera ADO.Net kod för dataåtkomst det är jobbigt och ensidigt.
- Microsoft har försett oss med ett **O/RM ramverk som kallas "Entity Framework"** för att **automatisera databasrelaterade aktiviteter** för ditt program.
- Microsoft har gett följande **definition** av Entity Framework:

  Microsoft ADO.NET Entity Framework är ett objekt / Relational Mapping (ORM) ramverk som gör det möjligt för utvecklare att arbeta med relationsdata som domänspecifika föremål, vilket tar bort det största behovet av att skriva dataåtkomst kod som utvecklare behöver vanligtvis skriva. I och med användningen av Entity Framework, så skapar utvecklare frågor med LINQ, som sedan hämtar och manipulerar data som starkt typade objekt. Entity Framework's ORM implementering tillhandahåller tjänster som ändringsspårning, identitetsupplösning, lazy loading, och fråge översättning så att utvecklarna kan fokusera på sin applikationsspecifika affärslogik snarare än dataåtkomst grunderna.

  Entity ramverk är ett objekt / Relational Mapping (O / RM) ramverk. Det är en förbättring av ADO.NET som ger utvecklare en automatiserad mekanism för åtkomst & lagring av data i databasen.

Vad är Entity Framework? (forts).

Entity ramverket är användbart i tre scenarier.

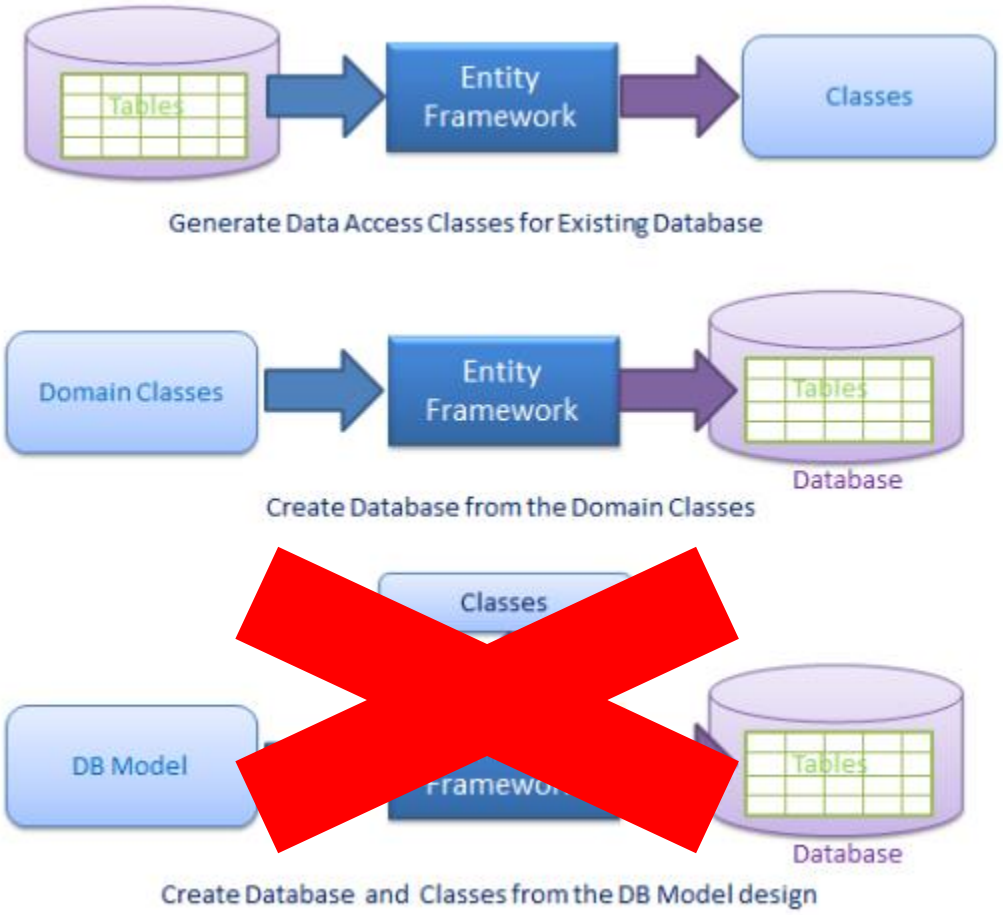
- **Först**, om du redan har befintlig databas eller om du vill designa din databas före dina andra delar av programmet.

- **För det andra**, du vill fokusera på dina domänklasser och sedan skapa databasen från dina domänklasser.

- **Tredje**, du vill designa ditt databasschema med den visuella designern och sedan skapa databasen och klasser.

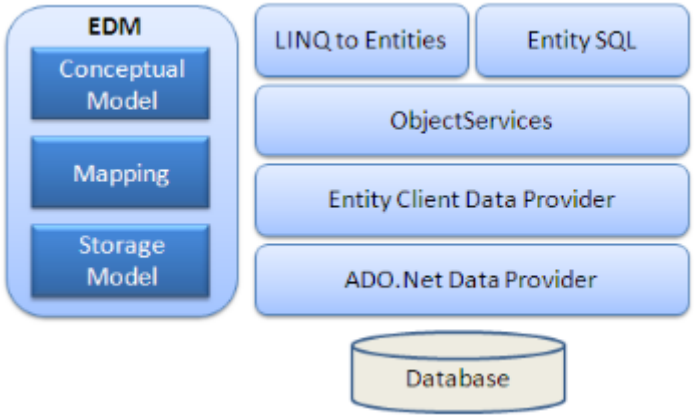
  Figur på nästa sida illustrerar ovanstående scenarier.

Vad är Entity Framework? (forts).



Entity Framework **arkitekturen**

Följande bild visar den övergripande arkitekturen för Entity Framework.



Olika **vägar** till Entity Framework?

- **Model to database** (depricated)
  - Vi bygger själva upp vår modell och utifrån den skapas databasen
- **Code first**
  - Vi skapar våra klasser och utfrån dem genereras databasen
- **Database first**
  - Utfrån en befintlig databas skapar vi vår modell

Hur hanterar vi **ändringar** i vår **Db**?

- Vi kan vilja **uppdatera** fält mm i vår Databas
- Dessa *ändringar* vill vi ska **reflekteras** i vår Databas
- Vi har **färdiga verktyg** i *Visual Studio* för **automatisera** detta
- Vi kan både **lägga till** *nya* tabeller och **uppdatera** *befintliga*

Hur hanterar vi **ändringar** i vår **Db** *(forts)*

- Kör alternativet ***Update model from database***
- **Bygg om projektet** för att ändringarna ska reflekteras i koden
- Om vi **ändrar typen** av *ett fält* i Databasen så kommer dessa **INTE** *att reflekteras i modellen*.
  - Fullt logiskt eftersom det kan ändra logiken i vårt program
  - Ex. Tänk t ex på ***string*** *som vi byter till* ***int***. Vi kan ha ha 50 olika ställen i vår kod som försöker använda just detta fältet som en sträng. **INGA** av dessa *skulle fungera om vi skulle göra den ändringen*.
  - Den ändringen måste vi göra **manuellt** *först i modellen* och *sedan uppdatera vår kod*

Enums

- Är en distinkt typ i c# som består av ett **set(samling) av konstanter,** kallad enumerator listan
- Per **Default** så *startar* den på 0 och ökar med 1
- En Enum **kräver inte** att vi deklarerar en *siffra* **men vi kan** göra det.
- Den underliggande typen är per **default**: ***int***

Extension metoder

- Vi kan skapa *hjälp metoder* som **kopplas till en specifik typ**
- Vi behöver då i vår kod **inte ha med** den **statiska klassens namn** utan kan **kalla på den direkt** *från den typen den implementerats på*
- T ex. **istället** för att skiva **StringUtils.ToDouble(myString)** kan vi skriva **myString.ToDouble()**
- För att det ska fungera måste vi ha med *den statiska klassens namn* i ett **using statement** i vår klass om extension klassen inte är med i vårt namespace

Extension metoder *(forts)*

- Vår klass ska vara en **statisk klass**
  - Så att vi kan kalla på den utan att skapa en instans av den
- **Metoden** vi *skriver måste också* vara **statisk**
- Vi använder ***this*-nyckelordet** för att göra en metod till en extension metod.
- Metoden kommer att bli en extension-metod till den **första typen** *vi deklarerar i vår metod signatur*

```
public static class PersonExtensions
{
    public static int CalculateAge(this Person person) {
```

IEnumerable

- IEnumerable är den som gör att vi kan göra en foreach-loop
  - Genom den metoden som heter GetEnumerator
- ICollection som ärver från IEnumerable ger ytterliggare funktionaliet
- IList som ärver av ICollection ger oss t ex Add metoden
- **List** -> **IList** -> **ICollection** -> **IEnumerable**
- **Linq** ger oss en uppsättning av **extension metoder** som vi kan *använda på IEnumerable*

Varför LINQ?

- Vi vill kunna **ställa frågor** till våra **Objekt/Listor/Arrays** för att få ut data
  - Objekten kan t ex vara en lista av strängar, en lista med Objekt eller kanske enititeter från Entity Framework
- För att göra detta kam man klassiskt skriv en **foreach/eller for-loop.**
- Vill vi ha vissa villkor får vi använda **if satser**
  - T ex *Bara personer vars namn startar på ”A”*
- Vill vi sedan t ex **sortera** vår resultat så får vi skriva *ytterligare kod*

Deferred Execution

- När vi skiver vår Linq fråga skriver vi vad som ska utföras. Vi har dock **INTE kört frågan ännu**.
- Frågan **körs** först när vi *använder* frågan
- Detta spelar stor roll i vad vi får tillbaka. Vi måste vara **försiktiga** så att vi får *tillbaka de resultat vi förväntar oss*
- Att frågorna körs först när vi kallar på dem kallas för **Deferred Execution**

Hur arbetar vi mot EF modellen i kod?

- **1.** Det första vi måste göra är **öppna** upp vårt **context**.
  - Det är här som vi arbetar mot vår databas
  - Saker som sker inom vårat context hjälper EF oss att hålla koll på.

- **2.** I vårt context så kan vi utföra olika arbeten
  - T ex: Lägga till, Ta bort och Ändra

- **3.** När vi gjort vad vi ska så kallar vi på SaveChanges() – som sparar ner det arbete som vi gjort

Skriva över SaveChanges()

- **SaveChanges()** metoden kommer att **spara ner** vårt *arbete vi gjort mot Enity Framework*

- Metoden **kommer** dock att **misslyckas** om vi **försöker spara ner saker till databasen som inte Db:n kan spara ner** t ex pga felaktigt format
  - T ex ett DateTime.MinValue(). *.NETs min value är mycket lägre än vad som tillåts i en MS SQL Databas.*

- Vi kan då **skriva över SaveChanges()-metoden** för att tillföra *egen validering/funktionalitet mm.*

Vilka ”varianter” av LINQ har vi?

- LINQ to Objects
- LINQ to XML
- LINQ to SQL
- LINQ to Entities

Entity Framework

- LINQ to Entities and Entity SQL
- Change tracking and identity management
- Serialization and data binding
- Connection and transaction management

DBSet

```
public class DbSet<TEntity> :
System.Data.Entity.Infrastructure.DbQuery<TEntity>
        where TEntity : class
    Member of System.Data.Entity
```

DBSet's är:

- En samling av en enda entitets typ
- En uppsättning metoder som: Add, Attach, Remove och Find
- Används med DbContext för att ställa fråge operationer mot databasen

Ladda relaterad data

Överväg en typisk modell för försäljningsinformation.

En kontakt är relaterad till en kund, en kund har kundorder, varje kundorder har rader, varje rad avser en produkt, varje produkt kommer från en leverantör som är också relaterad till en kategori. Kan ni föreställa er om du efterfrågar kontakter, och utan att det förväntas, hela innehållet i databasen returnerades - eftersom det var relaterat?

Ladda relaterad data (forts).

Detta kallas för deffered loading eller implicit deferred loading eller till och med (som det är mer känt som) lazy loading.

Från och med Asp.Net 4.0 så gör Entity Framework lazy loading som default.

```
var customers = from c in context.Customers select c;
foreach (var customer in customers)
{
    Console.WriteLine("{0} #Orders: {1}", customer.LastName, customer.Orders.Count());
}
```

Eager loading.

Eager loading tar en sträng in som parameter, strängen är namnet på den navigation propertyn som den skall hämta entities igenom.

Om vi tänker tillbaka till exemplet med kunder, orders produkter osv. Så skulle vi kunna plocka ut orderdetaljer genom orders enligt följande:

```
Var customers =
context.Customers.Include(”Orders.OrderDetails”).Include(”Adress
es”).ToList();
```

`Orders.OrderDetail` kallas en *query path* och när vi lägger till Include(”Adresses”) kallas det för *Chaining*

Mer om laddning

Om vi slår av Lazy loading enligt bilden nedan så kan vi ändå i speciella fall kringgå och ladda in flera relaterade attribut som den markerade kodsnutten i bilden.

```
using (var context = new SchoolDBEntities())
{
    //Disable Lazy loading
    context.Configuration.LazyLoadingEnabled = false;

    var student = (from s in context.Students
                   where s.StudentName == "Bill"
                   select s).FirstOrDefault<Student>();

    context.Entry(student).Reference(s => s.Standard).Load();
}
```

Explicit loading (forts).

- Till skillnad från lazy loading, finns det ingen tvetydighet eller risk för förväxling om när en fråga körs.
- För att göra så att du använder metoden load på relaterad entitets entry (vi anropar funktionen entry).
- För en en-till-många relation, anropa metoden load på Collection.
- Och för en ett-till-ett förhållande, anropa metoden load på Reference