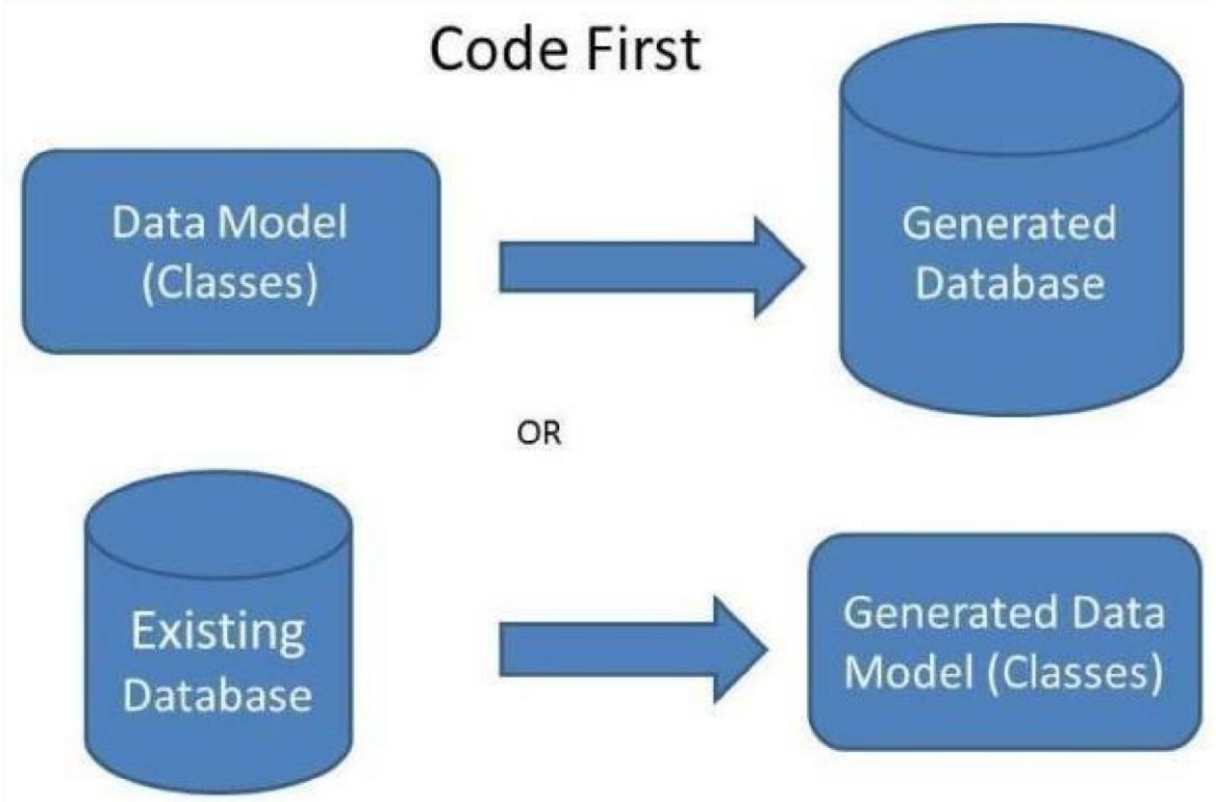
Av **Erik ”Zkall” Svensson**
@esdev.se

Entity framework: Code First databasmodellering



Vad är Entity Framework?

Ett förenklat verktyg att skapa databasmodeller. Jämfört med SQL kommandon så sparar du grymt mycket tid på att använda EF, när du ska spara/updatera/lägga till/ta bort något i databasen kan du göra det med 2-3 kodrader hellre än 10 + som SQL kommandon skulle kräva.

Vad innebär Code first?

Code first innebär att du skapar modelerna (classer) först och sedan skriver ett databas context för att komma åt allting ifrån. Vi dyker in i koden på igen gång så får ni snart en uppfattning av hur det fungerar

Modelerna (classerna)

Så det du börjar med när du gör code first är att skriva databas modelerna. En model skrivs så här:

```
namespace WebApplication1.Models
{
    [Table("Apor")]
    0 references
    public class ApaModel
    {
        0 references
        public virtual int Id { get; set; }
        0 references
        public virtual string namn { get; set; }
    }
}
```

[Table(”namn”)] Skrivs som parameter ovanför class namnet, det deklarerar att det ska skapas ett table vid namnet du anger. **public** krävs för att du ska komma åt klasserna från andra ställen i koden. **virtual** är till för ”Lazy Loading” – Inte nödvändigt och näst intill irrelevant för modeler och viewmodels.

public virtual int Id { get; set; } Används för att sätta en identifier key, du behöver inte skriva några parametrar utöver just den här raden, entity framework fattar att **int Id** är en primärnyckel och skapar den utan att fråga i databasen. OBS! Om ni inte har en Identifier deklarerad så kommer ni få error när ni compilar och kör koden.

classnamnModel där Model som avslut på class namnet är rekommenderat så att ni kan hålla koll på vad som är vad när ni jobbar med större projekt.

Databas context

När du har lagt till alla modeler är det dags att skapa ett context, det ser ut så här:

```
public class DB : DbContext
{
    1 reference
    public DB()
        : base("DefaultConnection")
    {
        Database.CreateIfNotExists();
    }

    0 references
    public static DB Create()
    {
        return new DB();
    }
}
```

För att du ska kunna refferera till dbcontext så behöver du installera Entity Framework i din app: install-package EntityFramework. Arvet ger oss det mesta så vi behöver inte deklarera så mycket mer än följande egenskaper:

1. Konstruktorn **DB() : base(”DefaultConnection”) { Database.CreateIfNotExists; }**

Det du gör där är att när du kallar på en DB class så skapar du database om den inte finns, behövs bara göra en gång så man bör egentligen inte ha den I konstruktorn. Om ni skippar den så skriver ni db.Database.CreateIfNotExists(); ifrån en controller t.ex.

2. Create metoden är där för att skapa klassen ehm? Den vet ja faktiskt inte om man behöver ha med, tänk inte på den.
3. **Public Dbset<ApaModel> Apor { get; set; }** Så skriver du för varje model du har skapat, där du deklarerar att det är en ApaModel, att den ska kommas åt från samlingsnamnet Apor och att den är get och set; Se nedanstående kod för referense:

```
3 references
public class DB : DbContext
{
    1 reference
    public DB()
        : base("DefaultConnection")
    {
        Database.CreateIfNotExists();
    }

    0 references
    public DbSet<ApaModel> Apor { get; set; }
    0 references
    public static DB Create()
    {
        return new DB();
    }
}
```

Hur kommer man åt databasen från koden

Du refererar till namnet av db classen (rekommenderat namn är DB då det är rätt uppenbart när du ska deklarera vad som är vad. ,så du skriver:

```
private DB db = new DB();
0 references
public ActionResult Index()
{
    var model = db.Apor.ToList();
    return View();
}
```

Sedan skriver du bara db.Pluralnamn av dbset du skapade och sen db.pluralnamn.add/remote/update/

Efter du har gjort en ändring måste du skriva db.SaveChanges(); annars sparas inte ändringarna.

Lägga till en model i databasen

```
private DB db = new DB();
0 references
public ActionResult Index()
{
    var temp = new ApaModel { namn="Adolf" };
    db.Apor.Add(temp);
    db.SaveChanges();
    return View();
}
```

Id behöver du inte skriva, utan det sköts per automatik.

Updatera en model i databasen

```
private DB db = new DB();
0 references
public ActionResult Index()
{
    var apan = db.Apor.First(x => x.namn == "Adolf");
    apan.namn = "Nisse";
    db.SaveChanges();
    return View();
}
```

Hämta först det du ska updatera, ändra sedan variablerna du vill ändra, och avsluta med db.SaveChanges();

Ta bort en model i databasen

```
private DB db = new DB();
0 references
public ActionResult Index()
{
    var apan = db.Apor.First(x => x.namn == "Adolf");
    db.Apor.Remove(apan);
    db.SaveChanges();
    return View();
}
```

Samma som förra exemplet, hämta objektet du ska modifieraa. Skriv sedan db.Apor.Remove(apan); Avsluta med db.SaveChanges();

Hämta flera

```
private DB db = new DB();
0 references
public ActionResult Index()
{
    var model = db.Apor.ToList();
    return View();
}
```

.ToList() för List > IEnumerable imo :P

Hämta flera med parametrar

```
private DB db = new DB();
0 references
public ActionResult Index()
{
    var model = db.Apor.Where(x => x.namn == "Nisse").ToList();
    return View();
}
```

.Where(x => x.namn == ”Nisse”) hämtar alla apor som heter Nisse. **X => X == x.namn** är lambda som jag går in på mer i ett annat dokument, för nu kan ni tänka på att det är så det funkar bara.