Entity Framework

Mål med lektionen!

- Förstå skillnaden mellan Lazy, Eager och Explicit loading

Vad lektionen omfattar

- Gå igenom Lazy, Eager och Explicit loading av data i EF.

Hur EF ”bygger” entiteterna

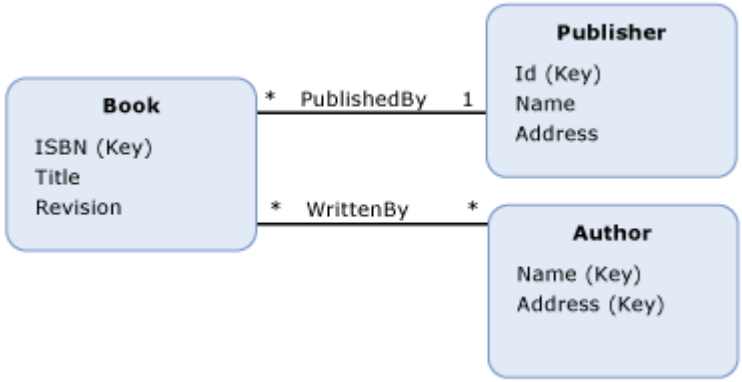
Om vi inte har några nycklar i en tabell så kommer inte EF att generera någon klass för denna, detta för att EF måste kunna göra sk Change Tracking, bygga relationer och merga query resultat.

EF använder sig av databasens primära nyckel för att skapa upp entitets nyckeln för modellen.

Där finns också en instans av en nyckel klass som skickas med varje hämtad instans av en entitet, denna klass är uppbyggd och genererad på ett lite speciellt sätt.

Hur EF ”bygger” entiteterna (forts).

En Entitetsnyckel objekt består av EntitySetName och EntityContainerName propertys och en array av en eller flera nyckel/värde par



Hur vi själva kan ändra nyckeln efteråt

Om vi vill ändra entitets nyckeln så kan vi antingen högerklicka på entitetens property/attribut och klicka för/av valet för ”Entity Key” eller högerklicka på vald property/attribut välja Properties och i properies fönstret välja ”false/true” för ”Entity Key”.

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());
}
```

Ladda relaterad data (forts).

Koden på föregående bild går igenom resultatet, så frågar den också efter information angående de relaterade orderna som finns. Men Orderna är inte returnerade med original frågan.

Detta är implicit loading av relaterad data.

Det man skall tänka på är att för varje gång som man kommer till Count() så kommer det att generera ytterligare en fråga på servern.

Tänkvärt om lazy loading

Om vi har en lista bestående av andra (relaterade) properties, då får vi en sk navigation property. Dessa kommer att bli ”public virtual”, om det skulle vara så att vi inte deklarerar sådana listor som virtuella så kommer contexten inte kunna göra lazy loading på/med dem.

Tänker kanske mest på de tillfällen då vi gör en sk code-first lösning, där vi skapar upp våra domänklasser själv.

Lite annat att tänka på ang lazy loading är att detta fungerar kanske inte superbra med serialisering eftersom detta kan göra att systemet tror att det mesta är relaterat och vi kommer att få ”hela” databasen.

Demo

Lazy loading

Lite mer att tänka på om lazy loading

Det finns ett stort prestanda övervägande här. Oavsett om du använder lazy-loading eller explicit laddar relaterad data för varje entitet, så tvingar koden fram en extra tur och retur till databasen, något som många utvecklare inte inser om de inte gör ”profiling” på databasaktiviteten.

Detta kan vara extremt ineffektivt och det kan också ge dig stora problem med ”IT-proffsen” i ditt företag. Med lazy loading inaktiverad, kan du ha viss kontroll över när den här extra resan görs.

Eager loading

Detta är den process varigenom en query för en typ av entitet laddar också närstående entiteter som en del av frågan. Eager loading uppnås genom att använda sig av ”Include” metoden.

Det betyder att den begärda relaterade datan returneras tillsammans med frågeresultat från databasen. Det finns bara en förbindelse som görs till datakällan och en större mängd data returneras i den ursprungliga frågan.

Till exempel om vi gör en query mot studenter, då kommer det bli en eager load mot enrollments. Studenterna och deras enrollments kommer att hämtas i en stor fråga/query.

Demo

Eager loading

Eager loading (forts).

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*

Lazy loading vs Eager loading

Vilket av dessa skall vi välja då?

Det ena har en massa turer till databasen och den andra laddar en massa data på en gång, vilket val vi än väljer så verkar det ju inte riktigt optimalt men det är så det är och hur vi än väljer så är det en liten balansgång som vi måste lära oss att gå. Valen kommer självklart att variera från tillfälle till tillfälle, men jag kan inte råda er till att välja det ena före det andra därför att det kommer att bero på situationen och systemets krav som gäller för stunden som kommer att avgöra vad som är bäst.

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();
}
```

Explizit loading

Även om vi stänger av vår lazy loading antingen i våra domänklasser genom att ta bort alla virtuella nyckelord för våra navigation properties eller om vi gör det med en kodrad i vår main metod.

```
context.ConfigurationLazyLoadingEnabled = false;
```

Så kan vi fortfarande ladda relaterade entiteter men det är lite vi skall tänka på för att kunna göra detta.

Explizit 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

Demo

Explicit loading

Lab 6

Obs denna labb är Obligatorisk och skall visas upp för mig.