## Tutorial Linq2Entities – Michiel Borkent – .NET Programmeren mei 2011

### geupdated 23 mei 2012 – oracle ondersteunt nu ook Entity Framework

In deze Tutorial gaan wij er vanuit dat je SQL Server Express lokaal op je systeem hebt geïnstalleerd. Voor het gebruik van .NET in combinatie met de Oracle-database van school~~, lees het document “DotNet in combinatie met Oracle”~~  lees de groene (extra) stappen.

Stappen voor Oracle ipv SQL Server zijn in groen weergegeven:

* installeer de nieuwste versie van ODAC (op het moment van schrijven versie 11.2.0.3.0): http://www.oracle.com/technetwork/developer-tools/visual-studio/downloads/index.html
* plaats in C:\Oracle\product\11.2.0\client\_1\Network\Admin een file tnsnames.ora met inhoud:

ORACLEHU =

(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = ondora01.hu.nl)(PORT = 8521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SERVICE\_NAME = cursus01.hu.nl)

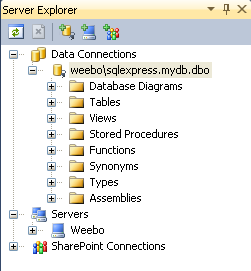
)

)

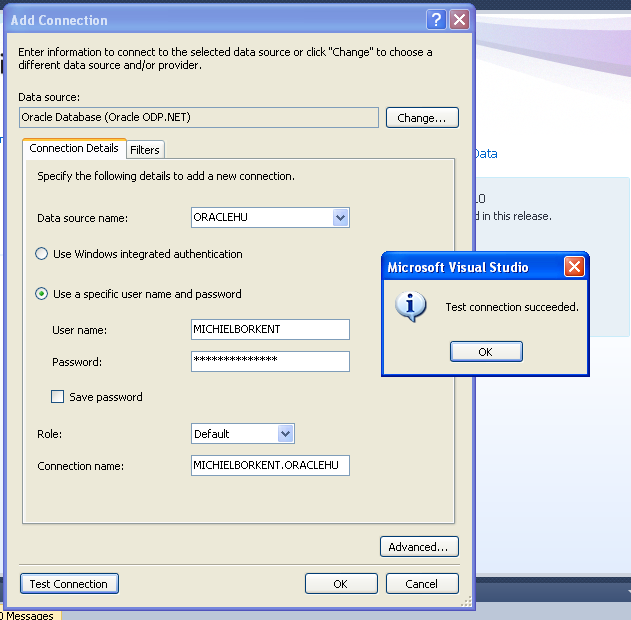
1. In Visual Studio, maak een nieuw project aan. Dit kan in ons geval een simpele Console-applicatie zijn.
2. Laat de Server Explorer zien middels View > Server Explorer

Voor SQL Server volg de stappen 3-6 en ga verder bij stap 9.

1. Onder Data Connections kies “Create New SQL Server Database” (rechtsklikken)
2. Bij Server Name, vul in “localhost\SQLEXPRESS” (hostname/databaseservername)
3. Kies bij New Database Name een passende naam voor je nieuwe database, in dit voorbeeld “mydb”
4. Onder Data Connections verschijnt nu een extra entry “<computernaam>\sqlexpress.mydb.dbo” met daaronder allerlei eigenschappen van de database:



Stappen 7 en 8 alleen voor Oracle.

1. Voor Oracle: Onder Data Connections kies Add Connection
2. 

Vul nu in het scherm bij data source name ORACLEHU in en je eigen username en password. Klik op test connection, dit zou succesvol moeten werken.

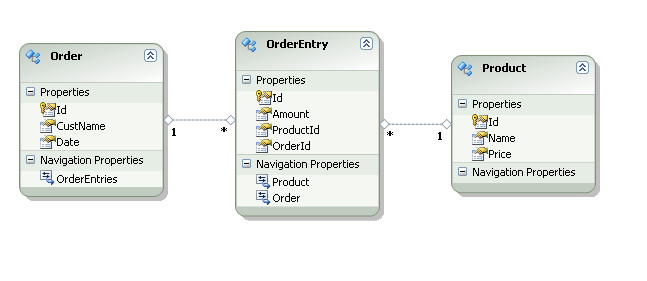
1. We gaan nu een zogenaamd “Entity Model” toevoegen aan onze applicatie. Met dit Entity-model kunnen we via Linq communiceren met de database. Voeg toe aan je project via “Add new item” een ADO.NET Entity Data Model. Kies weer een geschikte naam, in deze tutorial “mymodel.edmx” en bevestig je keuze.
2. Je kunt kiezen om een model te generen op basis van een database of met een leeg model te beginnen. In deze tutorial beginnen we vanaf scratch dus kiezen we “Empty Model”. Kies vervolgens Finish.
3. Visual Studio opent nu de “Design Surface” van je Entity Model. Hier kun Entities aanmaken die je kan gebruiken in je applicatie. We gaan een model creëren met de volgende Entities:

Order, OrderEntry, Product.

* Een Order is een bestelling met een String CustName (klantnaam) en DateTime OrderDate (datum van bestelling).
* Een OrderEntry is onderdeel van een Order en beschrijft hoeveel van één Product er is besteld. Een Order kan meerdere OrderEntries bevatten.
* Een Product heeft een String Name en een Double Price.

Rechtsklik op de “Design Surface” en kies “Add > Entity” en maak bovenstaande Entities. Laat bij alle Entities ook een Id genereren (standaard instelling).

Om attributen toe te voegen klik je op de Entity en vervolgens “Scalar Property”. Het type valt te wijzigen in de Properties Window. Standaard wordt als type String gekozen.

Associaties tussen Entities kun je aanmaken door op de “Design Surface” rechts te klikken en “Add > Association” te kiezen. Vervolgens kun je de twee bijbehorende Entities en de juiste multipliciteit en navigeerbaarheid kiezen; kies ook voor de opties “add foreign key properties”. Uiteindelijk krijg je een model wat er als volgt uit ziet. Bij Order moet je wel de OrderEntries op kunnen vragen en bij een OrderEntry moet je de bijbehorende Order en Product kunnen opvragen.

NB: Het is mogelijk om Entity models te maken die gebruik maken van inheritance en meer-op-meer-relaties. Deze worden door Entity Framework naar een database-model vertaald zoals jullie ook gezien hebben bij JDeveloper in het vak MDDT.

Oracle: vanwege een bug in visual studio kunnen de volgende wijzigingen niet via het property-window doorgevoerd worden, maar dit moet handmatig in de XML van het Entity Model gebeuren. Rechtsklik op de .edmx file en kies Open With -> XML Editor. Zoek de EntityType elementen op van de Entities die we zojuist hebben aangemaakt. Bij het Property-element van elke entity met Name=”ID” voeg een attribuut StoreGeneratorPattern toe met de waarde “Identity”. Dit zorgt ervoor dat Oracle zelf een nieuwe ids aanmaakt en niet het .NET-framework.  
  
figuur alleen van toepassing bij Oracle

1. Nu is het tijd om ook de tabellen te maken in de database op basis van de entities.

Voor Oracle moet je dan nog het volgende wijzigen:  
To generate the DDL scripts, open the **Properties**window of **mymodel**. Change the **Database Schema Name** to **<jouw gebruikersnaam van Oracle in HOOFDLETTERS!!!>** and select **SSDLtoOracle.tt** for the **DDL Generation Template**. Select **Generate Oracle via T4(TPT).xaml** in the **Database Generation Workflow property** to ensure table per type DDL will be generated.

Rechtsklik op de “Design Surface” en kies “Generate Database from Model”. Kies de juiste connectie naar de zojuist aangemaakte database en kies de optie “Yes, include the sensitive data…” . Deze optie zet inloggegevens voor je database in een element in het bestand App.Config. Als je dat niet doet moet je in je code de inloggegevens setten voordat je toegang krijgt tot de database. Kies Next.

1. Er wordt nu een DDL-script gegenereerd. Deze moet je opslaan, als default wordt er nu voorgesteld de bestandsnaam “mymodel.edmx.sql”. Kies Finish.
2. Het DDL-script wordt geopend. Het is nu zaak dit script uit te voeren.

Voor Oracle moet je dit script uitvoeren met een Oracle-tool zoals SQLDeveloper.

Voordat je dat doet, verander in elke tabel NCLOB in VARCHAR2(20) (of langer als je dat nodig vindt). NCLOBS kunnen niet goed vergeleken worden in de Linq-provider van Oracle (== of .Equals op Strings gaat mis). Bij VARCHAR2 gaat het wel goed.

Voeg handmatig voor elke tabel een sequence en een trigger toe, zodat bij het inserten in de tabel een nieuwe unieke primary key gegenereerd wordt. Met een tool als SQL Developer kun je rechtsklikken op een tabel en vandaar trigger kiezen en dan Create (PK from sequence).

Rechtsklik op het script en kies “Execute SQL”. Kies bij Connect de servername “localhost\SQLEXPRESS” en als authenticatie “Windows Authentication” (standaard instellingen). Bij het drukken op “Connect” wordt het script uitgevoerd.

Entity Framework heeft nu mappings gecreeërd van de Entities naar de Database. Het is nu mogelijk om vanuit je applicatie Entity-objecten te creeëren en te persisteren naar de database. LINQ speelt een belangrijke rol in het bevragen van de database.

1. Om gebruik te kunnen maken van het Entity Framework in onze code moeten we een referentie toevoegen. Rechtsklik op het project en kies Add Reference. Onder de tab .NET selecteer je System.Data.Entity.
2. Open de file Program.cs en voeg de volgende using statements toe:

using System.Data;

using System.Data.Common;

using System.Data.EntityClient;

using System.Data.EntityModel;

1. Bekijk de file mymodel.Designer.cs onder mymodel.edmx. Daarin zie je een zogenaamde “context” class en de verschillend Entity-classes. Via een “context”-class kunnen we ‘objecten’ verkrijgen die corresponderen met rijen in de database. Deze objecten kun je aanpassen en vervolgens kun je de veranderingen doorvoeren naar de database. Ook kunnen we nieuwe objecten creëren en die vervolgens persisteren naar rijen in de database. Een instantie van de context-class bevat een “wereld” van objecten die corresponderen met de “wereld” in de database.

public partial class mymodelContainer : ObjectContext …

1. In de file Program.cs gaan we nu een instantie van mymodelContainer aanmaken. Dit doen we in een using-block, zodat de resources (een databaseconnectie) die door mymodelContainer in gebruik zijn vrijgegeven worden aan het eind van het using-block. Zet de volgende code neer in de Main-methode:

using (mymodelContainer ctx = new mymodelContainer())

{

…

}

Op de puntjes gaan we nu wat objecten creeëren in persisteren naar de database:

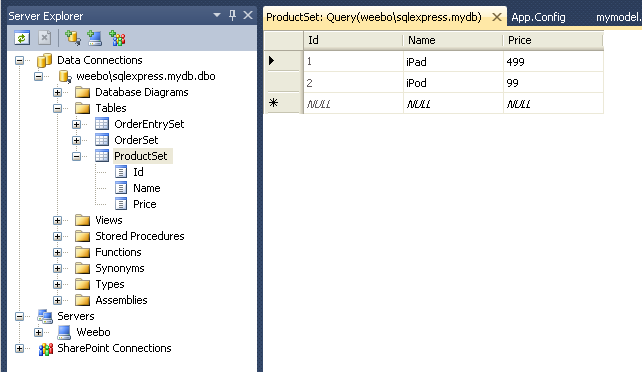
1. Product p1 = new Product { Name = "iPod", Price = 99 };

Product p2 = new Product { Name = "iPad", Price = 499 };

ctx.AddToProducts(p1);

ctx.AddToProducts(p2);

ctx.SaveChanges();

In de bovenstaande regels worden twee objecten van class Product aangemaakt. Deze class is afkomstig uit ons model mymodel. Vervolgens voegen we de objecten toe aan de context. De context heeft een “ObjectSet” van Producten waar alle referenties naar Producten in worden bewaard. Er is ook een tabel de database aangemaaktmet de naam Products. Vervolgens verzoeken wij de context om de veranderingen te bewaren. Dat is het moment dat er data naar de database gaat. Let op dat we hier zelf geen IDs hebben verzonnen voor de Producten. Deze worden door SQL Server (of Oracle) ingevuld. Controleer of er nu in de database twee rijen zijn toegevoegd in tabel Products (voorheen heette de gegeneerde tabel ProductSet, het plaatje is verouderd): 

Je kunt data in een tabel bekijken door te rechtsklikken en “Show Table Data” te kiezen.  
Voor Oracle geldt dat je de data in een tool zoals SQL Developer moet bekijken.

1. Commentarieer nu de regels uit stap 19 uit, anders worden er bij de volgende keer uitvoeren nog twee records toegevoegd in de Products-tabel. We gaan nu een LINQ-query loslaten op de data. Voeg de volgende regels toe in het using-block:

var products = from p in ctx.Products

select p;

foreach (Product p in products)

Console.WriteLine("Product met ID {0}, naam {1} en prijs {2}",

p.Id,

p.Name,

p.Price);

eventueel gevolgd door een:  
Console.ReadKey();

De LINQ-expressie haalt uit de context alle producten op zonder enige beperking. In de foreach-loop drukken we enkele eigenschappen van deze producten af:

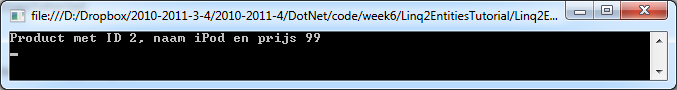


Natuurlijk kunnen we de LINQ-expressie aanpassen door bijvoorbeeld te beperken op prijs:

var products = from p in ctx.Products

where p.Price < 100

select p;



1. Omdat we niet alleen maar producten maar ook Orders en OrderEntries in ons systeem willen zetten gaan we die nu toevoegen. We gaan nu een Order maken voor de klant Peter. De datum van de order is vandaag en de Order omvat drie iPods en twee iPads. De Order zal worden opgeslagen in de database. Dat kan nu als volgt (voeg de code in het using-block toe en commentariëer de vorige code uit):

Product p1 = ctx.Products.Single(p => p.Name == "iPod");

Product p2 = ctx.Products.Single(p => p.Name == "iPad");

OrderEntry oe1 = new OrderEntry { Amount = 3, Product = p1 };

OrderEntry oe2 = new OrderEntry { Amount = 2, Product = p2 };

Order odr1 = new Order { CustName = "Peter", Date = DateTime.Today };

odr1.OrderEntries.Add(oe1);

odr1.OrderEntries.Add(oe2);

ctx.Orders.AddObject(odr1);

ctx.SaveChanges();

Toelichting bij Product p1 = ctx.Products.Single(p => p.Name == "iPod");

De methode Single selecteert een enkel resultaat gebaseerd op een lambda-expressie.

Die expressie moet dan beslist maar 1 resultaat hebben, anders treedt er een exception op.

Aangezien we maar 1 product hebben met de naam iPod of iPad gaat dit bij ons goed.

Toelichting bij OrderEntry oe1 = new OrderEntry{ Amount = 3, Product = p1 };

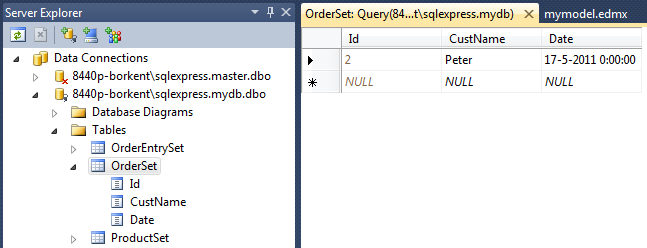
Een nieuw object oe1 van class OrderEntry wordt gecreëerd. Het Amount wordt op 3 gezet en het Product verwijst naar p1, die we zojuist opgehaald hebben uit de database.

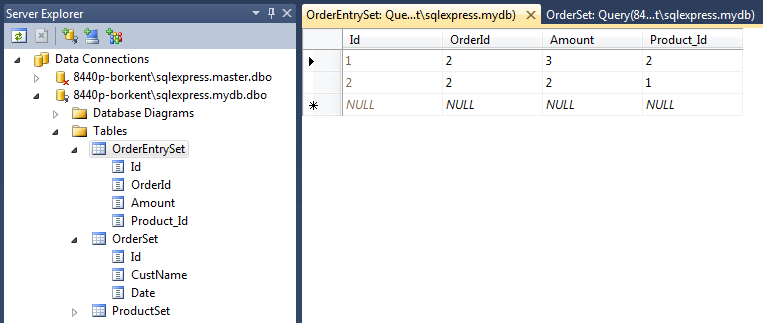
In de regel odr1.OrderEntries.Add(oe1); voegen we de OrderEntry oe1 toe aan de lijst van OrderEntries van order odr1.

odr1.OrderEntries is een collectie omdat een Order meerdere OrderEntries kan hebben.

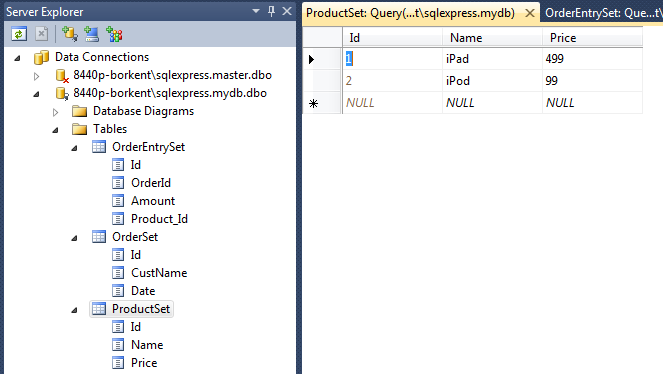
Als laatste stappen voegen we het object odr1 toe aan de Orders van de context en bewaren we de veranderingen. Hierbij worden ook objecten bewaard die aan odr1 “vast zitten”, namelijk oe1, oe2. p1 en p2 bestonden al in de database en worden dus niet gewijzigd.

Als we nu de data gaan bekijken in de database dan zien we het volgende.

In de tabel Orderset is een rij toegevoegd met CustName = Peter en de datum vandaag.



In de tabel OrderEntrySet zijn twee rijen toegevoegd die beide verwijzen naar de Order uit de vorige afbeelding met Id = 2. Op de eerste regel zie je dat er 3 bestelling van Product met id 2 zijn en op de tweede regel zie je dat er 2 bestellingen zijn van product 1.



Hier zien we nog even dat product met id = 1 een iPad is en product met id = 2 een iPod is.

1. In deze stap gaan we voor alle orders (op dit moment is het er één maar voeg er eventueel zelf nog wat toe) het totaalbedrag berekenen en afdrukken op het scherm. Dat kan als volgt. Vergeet niet de vorige code uit te commenten en voeg weer de volgende code tussen het using-block toe:

var query = from o in ctx.Orders

select new { Name=o.CustName,

Totals=(from oe in o.OrderEntries

select oe.Product.Price \* oe.Amount)};

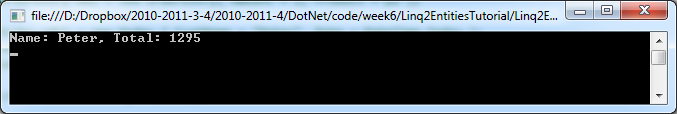
foreach (var result in query)

{

Console.WriteLine(result.Name + result.Totals.Sum());

}

In de vorige versie van deze tutorial werd de Sum methode meteen op Totals aangeroepen, maar dit ging bij Oracle niet goed: de vertaling van de Linq-query was voor de Oracle-database niet te begrijpen. Vandaar dat nu eerst de totalen worden verzameld. Vervolgens passen we de IEnumerable-versie van Sum toe op de lijst. Het optellen wordt dus in .NET gedaan en niet in de database. Wees over het algemeen zeer voorzichtig met het gebruiken van .NET-functies in linq-queries die naar de database gaan. De vertaling gaat niet altijd zomaar automatisch.



In de select van de LINQ-query creëren we een zogenaamde anonymous type. Dit is een class zonder naam en bevat hier twee velden, namelijk Name en Totals. De LINQ-expressie bevat dus als resultaat een IQueryable (we hebben met databases van doen) van dit anonymous type. De Name maken we gelijk aan CustName van Order en Totals maken we gelijk aan de lijst van prijzen van de producten die we via de OrderEntry-objecten kunnen vinden vermenigvuldigd met het aantal wat besteld is.

We kunnen door de resultaten heenlopen zonder dat we het type kennen van de resultaten (een anonieme class heeft geen naam), door middel van het var-keyword. We weten in elk geval dat de resultaten de velden Name en Totals bevatten, deze hebben we namelijk zelf in de LINQ-expressie genoemd. Op bladzijde 283 van het boek vindt je meer over het gebruik van anonymous types in LINQ-queries.

Verder zien we nog dat we de extension method Sum hebben gebruikt. Sum werkt op IEnumerable<double> maar ook bijvoorbeeld op IEnumerable<int>. Omdat we in de binnenste LINQ-expressie doubles selecteren (Price van Product is een double) wordt het resultaat een IEnumerable<double>. Als we dus op deze LINQ-expressie de methode Sum aanroepen krijgen we de som van alle prijzen die we hebben geselecteerd. Het volgende kleine voorbeeldje illustreert de werking van Sum.

double[] prices = { 1.0, 2.3, 3.4 };

double totalprice = prices.Sum();

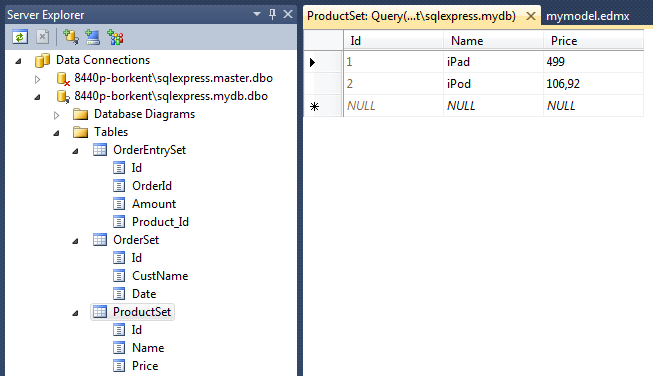
1. Het updaten van bestaande data is heel eenvoudig. Stel dat we de prijs van de iPod met 8% willen verhogen. Dan selecteren we het bijbehorende object, veranderen de prijs en bewaren de veranderingen in de context.

Product p1 = ctx.Products.Single(p => p.Name == "iPod");

p1.Price = p1.Price \* 1.08;

ctx.SaveChanges();

NB: We hoeven het object niet aan de context toe te voegen, omdat we het object namelijk uit de context afkomstig is.



1. Zaken verwijderen gaat net zo eenvoudig. Echter, als we dit voorbeeld proberen uit te voeren krijgen we een exceptie:

Product p1 = ctx.ProductSet.Single(p => p.Name == "iPod");

ctx.ProductSet.DeleteObject(p1);

ctx.SaveChanges();

omdat de referentie-contraints geschonden worden. Er verwijst namelijk een OrderEntry naar dit product. We zullen dan eerst deze OrderEntry moeten verwijderen, voor we de iPod kunnen verwijderen. Het volgende voorbeeld zal wel goed gaan:

Product p1 = ctx.Products.Single(p => p.Name == "iPod");

IQueryable<OrderEntry> orderentries =

(from oe in ctx.OrderEntries

where oe.Product.Name == p1.Name

select oe);

foreach (OrderEntry oe in orderentries)

ctx.OrderEntries.DeleteObject(oe);

ctx.Products.DeleteObject(p1);

ctx.SaveChanges();

EINDE