# .NET in combinatie met Oracle-database

**Update 21-5-2012**: Oracle en Entity Framework zou nu moeten werken volgens deze tutorial:

<http://download.oracle.com/oll/obe/EntityFrameworkOBE/EntityFrameworkOBE.htm>

**Update 23-5-2012:** Het document Tutorial Linq2Entities.docx is nu bijgewerkt met ondersteuning voor Oracle.

Michiel Borkent – 31-5-2011

### Installatie Oracle-software voor .NET

Oracle werkt sinds enige tijd aan ondersteuning voor Entity Framework. Echter deze ondersteuning is nog in een uiterst beta-stadium. Daarom is het op dit moment aan te raden om gebruik te maken van de “klassieke manier” van data access als je gebruik wil maken van Oracle.

Om Visual Studio geschikt te maken om te kunnen ontwikkelen in combinatie met Oracle moet je de volgende software installeren (zet hierbij Visual Studio uit!):

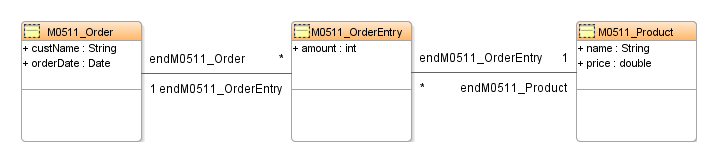
## ODAC 11.2 Release 3 (11.2.0.2.1) with Oracle Developer Tools for Visual Studio

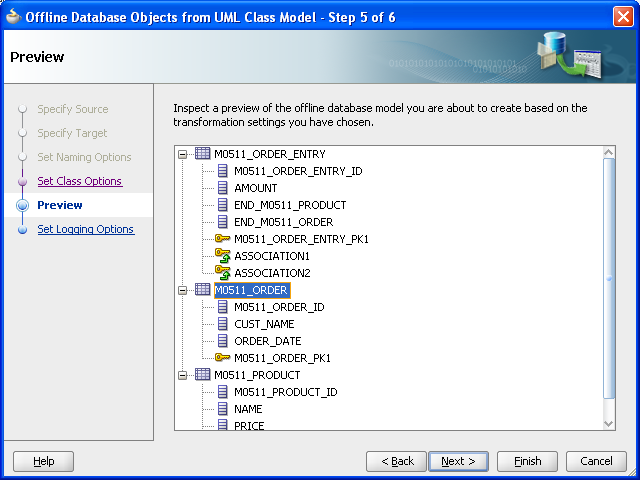
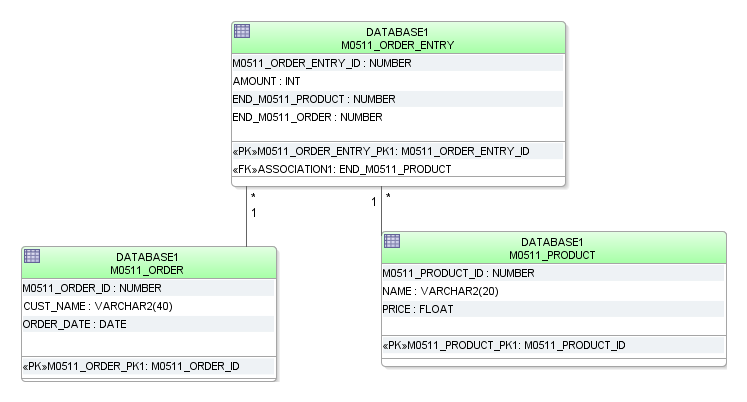
<http://www.oracle.com/technetwork/database/windows/downloads/index-101290.html>

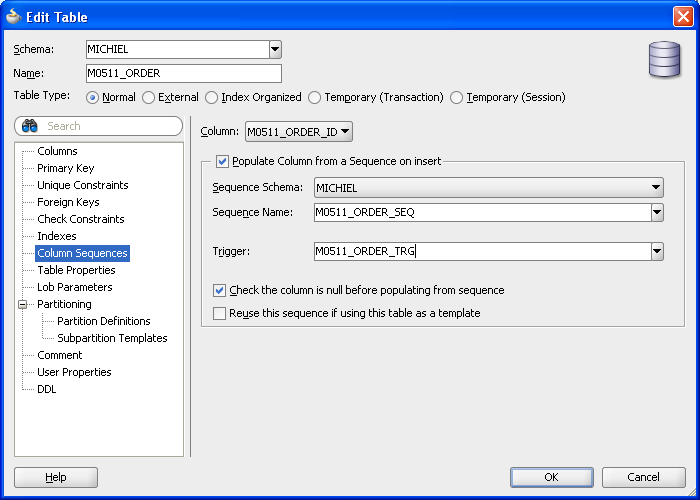
Kies bij de installatie voor ODAC for **Oracle Client (dus niet Server).**

Uitgebreide documentatie vind je op: **Documentatie:** <http://download.oracle.com/docs/cd/E11882_01/win.112/e18754/toc.htm>

### Stappenplan ontwikkelen .NET in combinatie met Oracle

In het vak MDDT hebben we gezien hoe we van een domeinmodel naar een DDL-script kunnen komen. We begonnen toen in JDeveloper met een UML-model en zetten dit via een transformatie om naar een Database Diagram en vervolgens naar een DDL. Die DDL voerden we dan uit in een schema, zodat er daadwerkelijk iets aangemaakt werd in de database. Zie daarvoor ook les 2 en opdracht 2 van het vak MDDT. Van dat principe kunnen we nu ook gebruik maken om toch nog redelijk dicht bij het voorbeeld te blijven uit het eerste gedeelte van dit document. Maar als je liever met de hand je database wil creëren is dat natuurlijk ook prima.  
In het volgende stappenplan staat beschreven hoe je van een model naar een werkend Visual Studio-programma dat gebruik maakt van een Oracle-database komt. Als je het met de hand doet, kun je dus een aantal stappen overslaan maar moet je zelf de DDL schrijven.  
  
1) Maak een nieuwe applicatie in JDeveloper en selecteer daarbij de technologieën UML en Database (Offline).   
  
2) Voeg aan de applicatie een nieuw class diagram (File -> New, onder Categories UML en dan Class Diagram kiezen)  
  
3) Voeg de gewenste classes met hun attributen toe (het type van de attributen wordt niet altijd goed getransformeerd, dus dat moeten we later in het database diagram nog aanpassen):  
(Ik laat de classes beginnen met een prefix zodat ik later nog weet bij welk project de genereerde tabellen horen, verder heeft dit geen betekenis)  
  
4) Selecteer alle classes en via kies rechtermuisknop Transform -> New Diagram -> UML to Offline Database Objects

5) Controleer of de transformatie naar wens zal verlopen. Let erop dat bij de Order\_Entry-tabel er een product\_id en een order\_id is. Als er een order\_entry-id bij Product wordt opgeslagen is dat niet gewenst, een product moet namelijk onafhankelijk van een order\_entry worden opgeslagen. Als alles klopt, kies dan voor Finish.    
  
6) Controleer het Database Diagram en corrigeer de datatypes naar wens (int, float, etc). Vergeet niet per tabel een “column sequence” voor de primary keys te maken, zodat de primary keys automatisch worden opgehoogd bij het aanmaken van nieuwe records. Kijk daarvoor bij de properties van de tabellen.

  
  
7) Kies Synchronize with Database - > Generate To -> SQL Script, zorg dat “Include depencies” aangevinkt staat (anders worden er geen triggers en sequences aangemaakt) en genereer het DDL-script. Controleer het script op fouten. Als alles OK is voer het script dan uit je in de schema (selecteer de tab SQL Worksheet en de juiste connectie).

8) De database is nu gereed voor gebruik. Nu is het tijd om in Visual Studio verbinding te leggen met de database en wat zaken toe te voegen en op te halen. Maak daartoe een Console-applicatie en voeg een referentie toe naar Oracle.DataAccess. Dit is een component wat is toegevoegd door de Oracle-software die je hebt geïnstalleerd.

9) Voeg bovenaan je programma de volgende regels toe:

using Oracle.DataAccess.Client;

using Oracle.DataAccess.Types;

using System.Data;

10) Om verbinding te kunnen leggen met de Oracle-database moeten we een connection-string opbouwen. Normaal gesproken staan er een aantal Oracle-gegevens in een bestand genaamd tnsnames.ora, maar we zullen het voor nu even voluit in de connection string zetten:

String connectionString = String.Format(@"DATA SOURCE=(DESCRIPTION=(ADDRESS\_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=ondora01.hu.nl)(PORT=8521)))(CONNECT\_DATA=(SERVER=DEDICATED)(SERVICE\_NAME=cursus01.hu.nl)));PASSWORD={1};PERSIST SECURITY INFO=True;USER ID={0}",

"…", // hier je username,

"…" // hier je password

);

Zoals je ziet staan er gegevens in zoals de hostname en poort waar de database te vinden is. Vul op de puntjes je username en password in voor je Oracle-schema.

11)

Om een connectie te maken voer je het volgende uit (aan het eind van het programma zullen we de verbinding weer afsluiten):  
OracleConnection conn = new OracleConnection(connectionString);

conn.Open();

12) We gaan nu een product toevoegen in de database. Omdat we het id willen weten van het product wat we hebben toegevoegd, voegen we een parameter toe in het stukje PL/SQL. Het id wordt dan teruggegeven in de parameter. Dit id hebben we nodig bij het maken van een OrderEntry. De methode ExecuteNonQuery voert een query uit waarbij je geen resultaten verwacht, zoals bij een select-statement.

OracleCommand cmd = conn.CreateCommand();

cmd.CommandText = "insert into M0511\_PRODUCT (NAME,PRICE) values('iPod',200) returning M0511\_PRODUCT\_ID into :p1";

cmd.Parameters.Add("p1", OracleDbType.Decimal, ParameterDirection.Output);

cmd.ExecuteNonQuery();

OracleDecimal idDec = (OracleDecimal)cmd.Parameters["p1"].Value;

int productId = idDec.ToInt32();

13) Vervolgens voegen we een Order toe, hiervan moeten we ook het id onthouden, zodat we later nog OrderEntries kunnen toevoegen die verwijzen naar deze Order en naar het Product wat we zojuist hebben toegevoegd.

cmd.CommandText = "insert into M0511\_ORDER (CUST\_NAME,ORDER\_DATE) values ('Michiel', TO\_DATE('2011/05/31 20:55:00', 'YYYY/MM/DD HH24:MI:SS')) returning M0511\_ORDER\_ID into :p1";

cmd.ExecuteNonQuery();

idDec = (OracleDecimal)cmd.Parameters["p1"].Value;

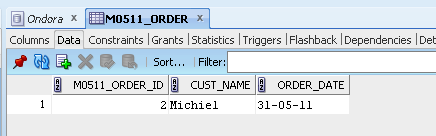
int orderId = idDec.ToInt32();

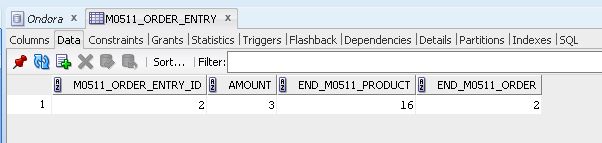
14) Nu kunnen we een OrderEntry toevoegen bij de Order en het Product. Als laatste sluiten we de verbinding.

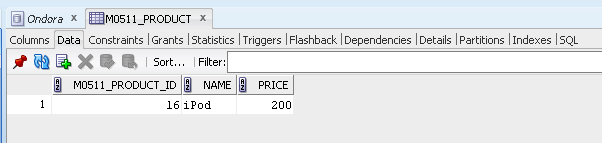
cmd.CommandText = String.Format("insert into M0511\_ORDER\_ENTRY (AMOUNT,END\_M0511\_PRODUCT,END\_M0511\_ORDER) values ('3',{0},{1})", productId, orderId);

cmd.ExecuteNonQuery();

conn.Close();





  
15) Het zou nuttig kunnen zijn om het totaalbedrag van de zojuist gecreëerde order te kunnen berekenen. Dat kun je met het volgende stukje code doen. De orderId is afkomstig van de eerder toegevoegde order. Deze voegen we ook weer in het select statement toe zodat we kunnen laten zien hoe je meerdere velden kunt opvragen in C#.

cmd.CommandText = String.Format("select OE.amount \* P.price AS TOTAL, {0} AS ID from M0511\_ORDER\_ENTRY OE, M0511\_PRODUCT P WHERE OE.END\_M0511\_ORDER = {0} AND OE.END\_M0511\_PRODUCT = P.M0511\_PRODUCT\_ID", orderId);

OracleDataReader reader = cmd.ExecuteReader();

16) De volgende stap is het uitvragen van de resultaten van het select-statement.

Omdat er meerdere rijen als resultaat kunnen worden opgeleverd moeten we in een loop de resultaten gaan opvragen. Dat doe je als volgt:

OracleDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

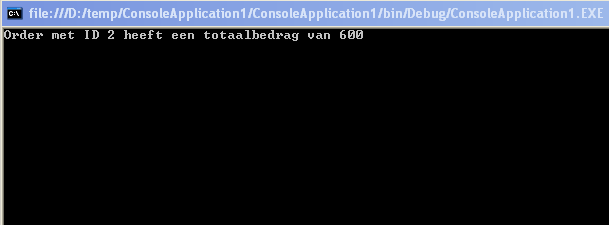
object total = reader[0];

object id = reader[1];

Console.WriteLine("Order met ID {0} heeft een totaalbedrag van {1}", id, total);

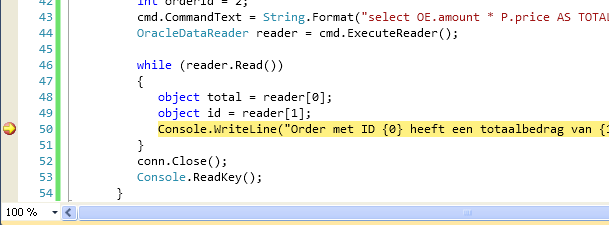
}

We voeren nu niet de query uit met ExecuteNonQuery maar met ExecuteReader. Deze methode levert een OracleDataReader op die de resultaten bevat van het select-statement. Vervolgens kunnen we door de resultaten heenlopen door het herhaald aanroepen van de Read-methode. Deze methode zorgt ervoor dat er naar de eerstvolgende rij wordt gesprongen en levert false op als er geen volgende rij meer is. Met reader[i] vraag je het i-de element op van de huidige rij. Omdat we hebben gezegd “select OE.amount \* P.price AS TOTAL, {0} AS ID from M0511\_ORDER\_ENTRY OE” is total het 0-de element van reader, en het id het 1-ste element. Reader gedraagt zich hier als een geindexeerde collectie, dat komt omdat C# zogenaamde “indexers” ondersteunt waardoor gewone objecten zich kunnen gedragen alsof ze een array zijn (meer hierover is in je boek te vinden). Je ziet dan het volgende resultaat op het scherm, slechts één regel, omdat we ook maar één order hebben toegevoegd:

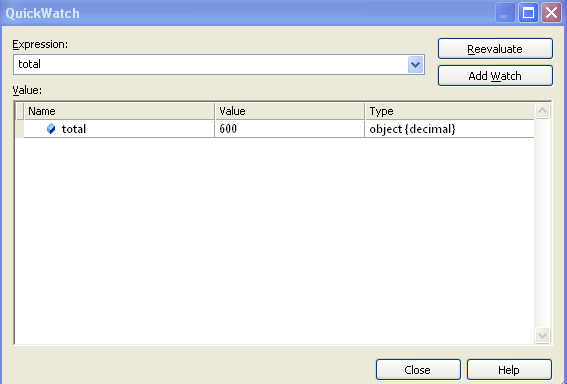


reader[i] geeft trouwens iets terug van type object, die je nog moet casten naar een bepaald type zoals String of Int32, het zogenaamde run-time type. Maar omdat ik niet precies wist welke types erachter verscholen gingen heb ik de volgende manier gebruikt om daar achter te komen:

Zet een breakpoint op regel 50 in onderstaand plaatje en run het programma:



Als het programma bij regel 50 is aangekomen kun je lokale waarden bestuderen en het run-time type zien. Rechtsklik op de variabelenaam total en selecteer “Quickwatch”. Je ziet dan het volgende:



We weten nu dus dat total het run-time type decimal heeft. Het zou dus moeten kunnen om dit object naar een decimal te casten. Hetzelfde kun je doen voor de variabele id die ook een decimal blijkt te zijn.

Het volgende werkt dan dus ook:

while (reader.Read())

{

decimal total = (decimal)reader[0];

decimal id = (decimal)reader[1];

Console.WriteLine("Order met ID {0} heeft een totaalbedrag van {1}", id, total);

}

Op vergelijkbare manier kan je Strings, etc uit je selecties C# binnenhalen. EINDE