

Copyright © 2015 by The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House
427 S. 4th Street #300
Louisville KY 40202

Entity Framework Database First

.NET Cohort

Coding Bootcamp

Lesson Goals

- What's an ORM?
- Learn to install Entity Framework from NuGet
- Build a simple model using the wizard
- Load some data from the database

Impedance Mismatch

- The problem many OO developers face is that object-oriented relationships do not always cleanly map to database structures.
- Thus, there is often a lot of plumbing work to be done to translate table schema to our objects.
- This code is tedious and error prone.

ORM – Object Relational Mapper

- An ORM, like Entity Framework, is designed to automatically generate classes and plumbing to map code objects to database schema and manage the relational mapping between them.
- Entity Framework 5.0 can start with an existing database, an object model, or code first.
 - ADO.NET, the .NET data frameworks, runs under the covers

Strengths and Weaknesses of ORM

Strengths

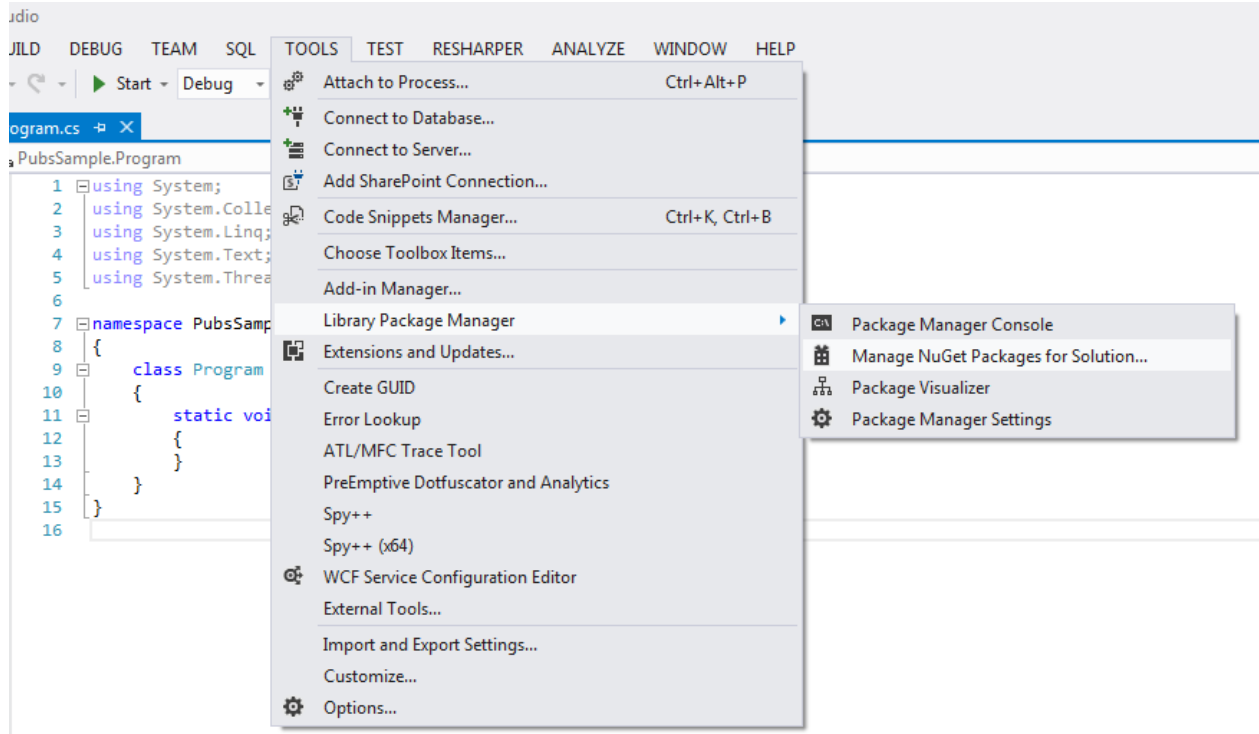
- Quickly able to generate data classes and get data into your application
- In simple cases, knowledge of SQL and database schema is not necessary to be effective
- Things like transaction management are much easier

Weaknesses

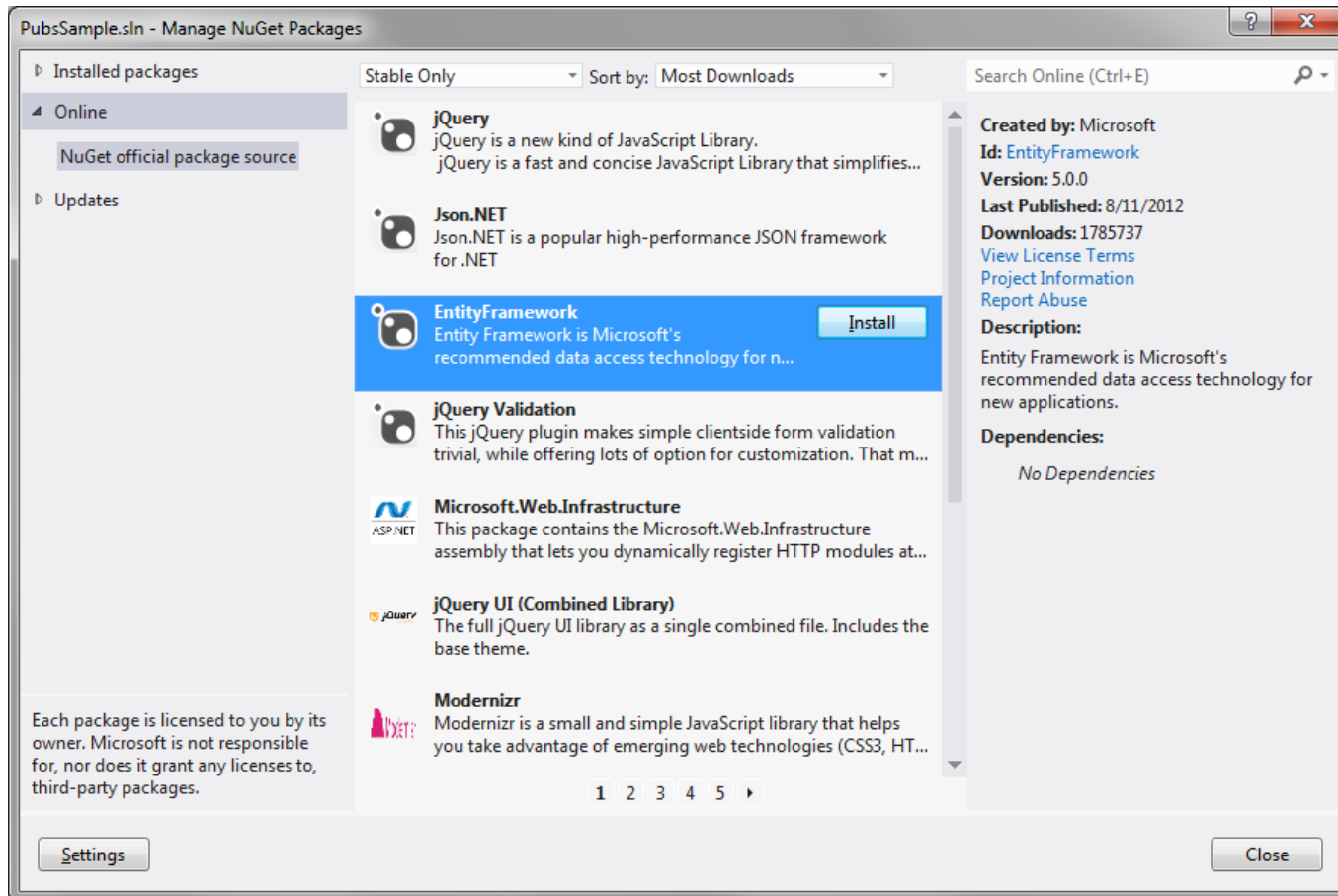
- Does not scale well, making it challenging for high-performance scenarios
- Getting down to the query level for granular control is difficult
- Database administrators tend to hate it because it makes performance optimization more difficult
 - Some queries and statements generate awful code.

Adding Entity Framework

We can get the latest version of the Entity Framework from NuGet



NuGet has all sorts of useful goodies; you can search in the upper right and install a package by hitting the Install button.



Next Steps

1. Add a model to the project
2. Use the Entity Data Model Wizard
3. Edit the model
4. Write the client code

Every database-first Entity Framework app follows these same steps.

Add a model

- To add a model to the project:
 - Right-click the project and choose Add New Item
 - In the Add New Item dialog, click on the Data template section on the left and choose ADO.NET Entity Data Model
 - Give it a name (in our case, we're going to use Northwind's database, so we will call it Northwind.edmx)

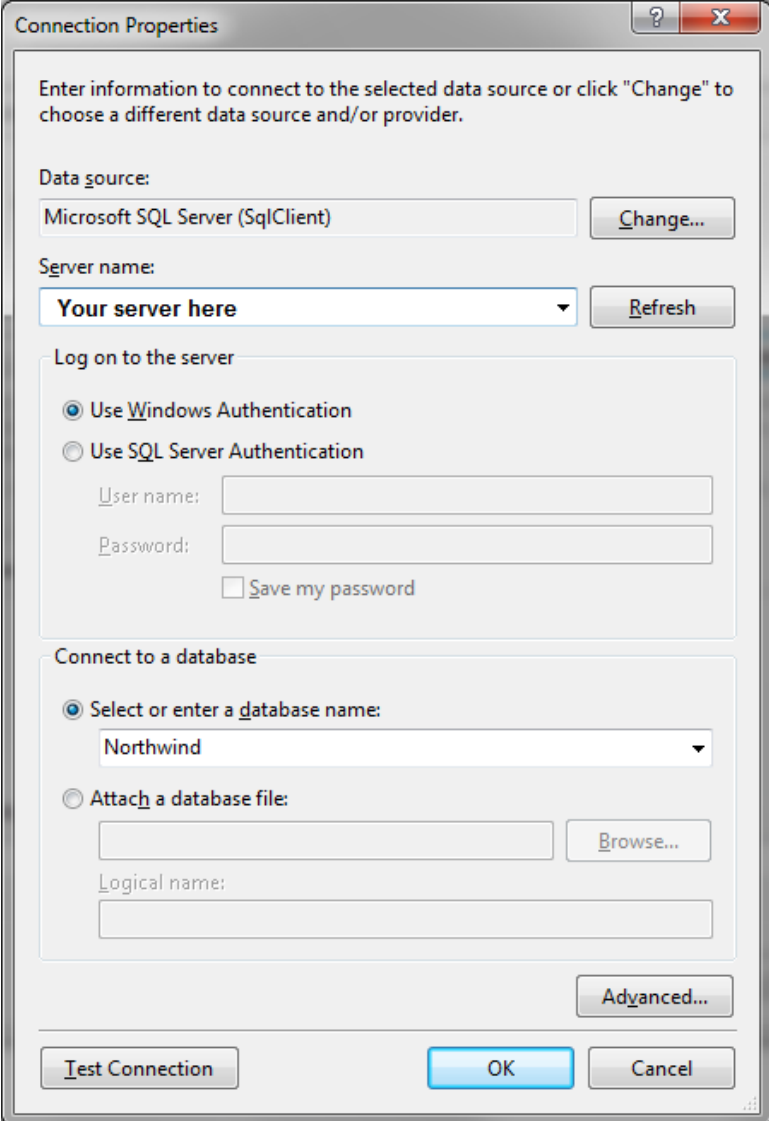
Entity Data Model Wizard

- Adding the model sends us into the wizard:
 - Choose Generate From Database
 - Set up a connection
 - Choose which database objects you want to bring into the model

Setting up a Connection

Generally you will use “localhost” as the server when developing locally.

Otherwise, ask your database administrator.



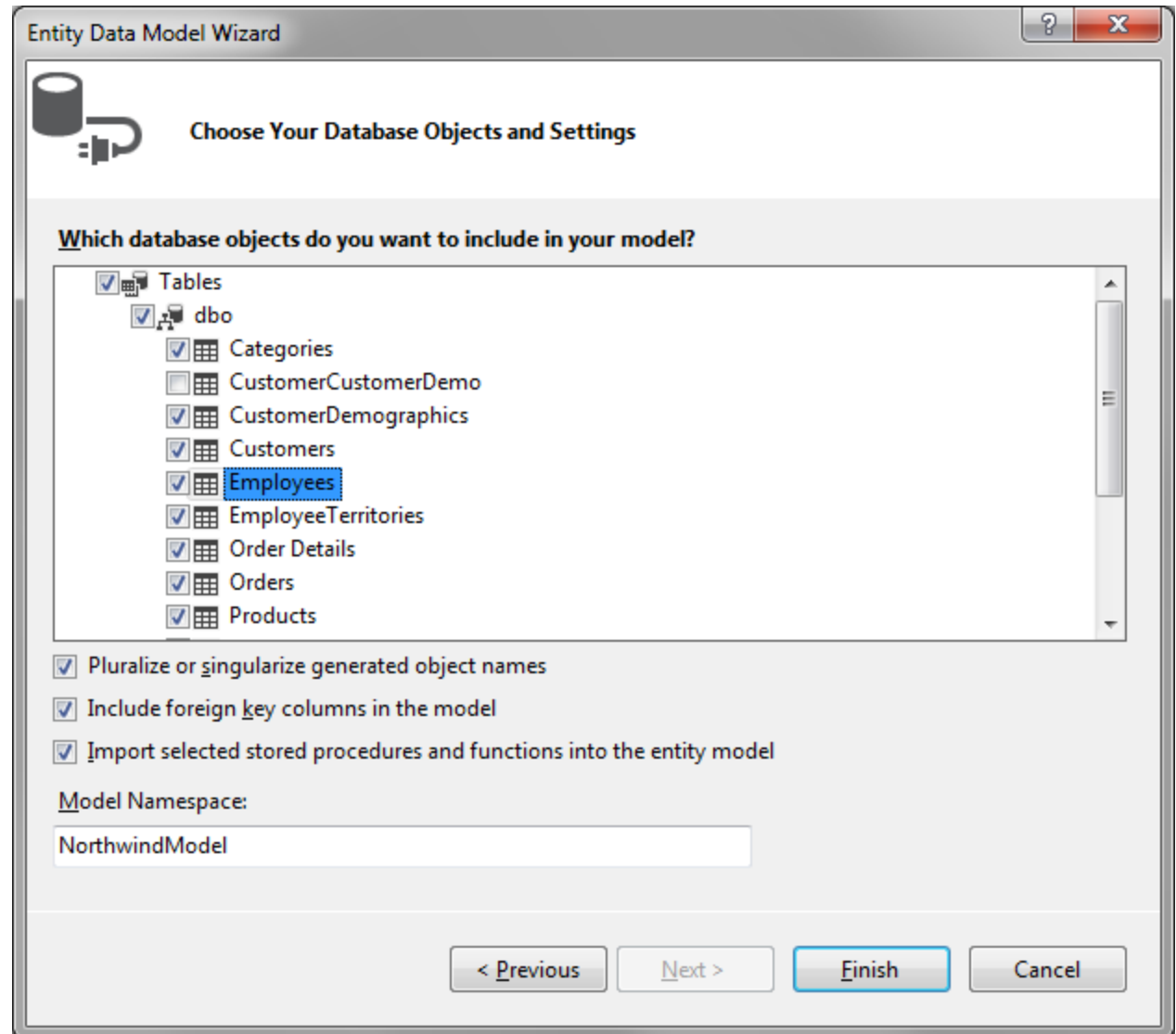
The screenshot shows the 'Connection Properties' dialog box with the following settings:

- Data source:** Microsoft SQL Server (SqlClient) [Change...]
- Server name:** Your server here [Refresh]
- Log on to the server:**
 - ☒ Use Windows Authentication
 - ☐ Use SQL Server Authentication
 - User name: []
 - Password: []
 - ☐ Save my password
- Connect to a database:**
 - ☒ Select or enter a database name: Northwind []
 - ☐ Attach a database file: [] [Browse...]
 - Logical name: []

Buttons at the bottom: Test Connection, OK, Cancel, and an Advanced... button.

Choosing Objects

For now, we will simply get some of the tables.



The image shows the 'Entity Data Model Wizard' dialog box, specifically the 'Choose Your Database Objects and Settings' step. The title bar reads 'Entity Data Model Wizard'. Below the title bar is a database icon and the text 'Choose Your Database Objects and Settings'. The main question is 'Which database objects do you want to include in your model?'. A tree view shows 'Tables' selected, with 'dbo' expanded. Under 'dbo', several tables are listed with checkboxes: 'Categories' (checked), 'CustomerCustomerDemo' (unchecked), 'CustomerDemographics' (checked), 'Customers' (checked), 'Employees' (checked and highlighted with a blue selection box), 'EmployeeTerritories' (checked), 'Order Details' (checked), 'Orders' (checked), and 'Products' (checked). Below the tree view are three checked options: 'Pluralize or singularize generated object names', 'Include foreign key columns in the model', and 'Import selected stored procedures and functions into the entity model'. At the bottom, there is a 'Model Namespace:' label and a text box containing 'NorthwindModel'. The bottom right corner has four buttons: '< Previous', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ Tables
 - ☒ dbo
 - ☒ Categories
 - ☐ CustomerCustomerDemo
 - ☒ CustomerDemographics
 - ☒ Customers
 - ☒ Employees
 - ☒ EmployeeTerritories
 - ☒ Order Details
 - ☒ Orders
 - ☒ Products

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

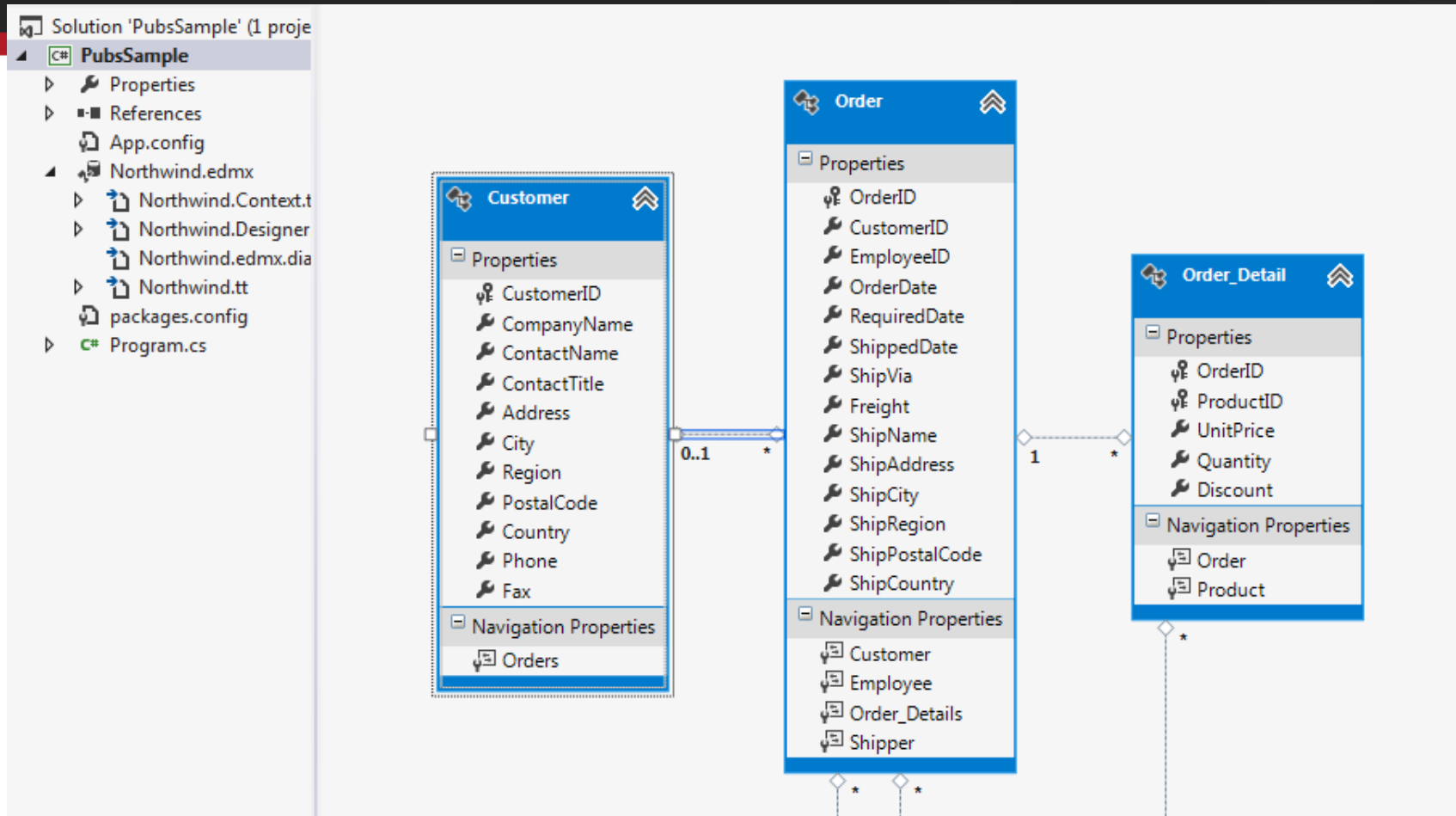
☒ Import selected stored procedures and functions into the entity model

Model Namespace:

NorthwindModel

< Previous Next > Finish Cancel

Ta da!



Edit the Model

- We can adjust names of model objects
- For example, I don't like Order_Detail and Order_Details as a navigation property, let's click on them in Order and Order_Detail and take out that underscore.

Write Some Code

- Now that we have a model and entity context in place, we can use the generated class (context) and LINQ to query the database.
- Let's write a query to list out all the customer in the USA to the console...

Getting USA Customers

```
class Program
{
    static void Main(string[] args)
    {
        GetUSACustomers();
        Console.ReadLine();
    }

    private static void GetUSACustomers()
    {
        using (var context = new NorthwindEntities())
        {
            var usaCustomers = context.Customers.Where(c => c.Country == "USA");

            foreach (var customer in usaCustomers)
            {
                Console.WriteLine("{0,-35} {1} {2}", customer.CompanyName, customer.Phone, customer.Country);
            }
        }
    }
}
```

Changing Data

- We can manipulate the collections in our database context and then call `SaveChanges()` to apply all of the changes to the database.

```
private static void AddRegion()
{
    using (var context = new NorthwindEntities())
    {
        Region newRegion = new Region();
        newRegion.RegionDescription = "My New Region";

        context.Regions.Add(newRegion);
        context.SaveChanges();
    }
}
```

Conclusion

- With database-first development, we can be up and running in a few minutes.
- A downside on the security aspect is that the application needs permission to access tables directly. We will demonstrate binding to stored procedures later.