<Entity Framework, MyBatis.NET, NHibernate 강좌 리스트>

목차

[Entity Framework 강좌] 01. Entity Framework 들어가기

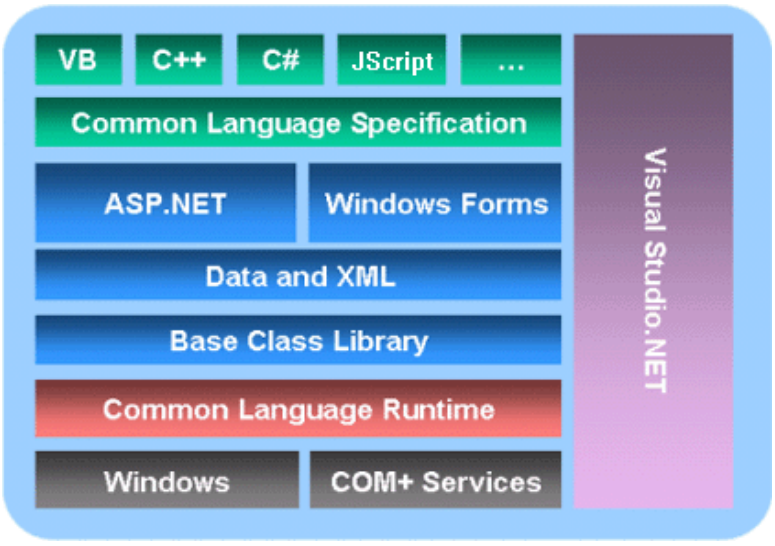
Entity Framework 들어가기

최근 닷넷계에서 화두 되고 있는 분야 중에 하나가 오픈 소스일 것이다. 우리는 그 동안 닷넷 프레임워크에서 제공되는 Base Class Library 기반에 어플리케이션을 잘 구현해왔다. 하지만 좀 더 효율적으로 좀 더 유연하게 프로젝트를 수행하기 위해 자바처럼 여러 프레임워크의 장점들을 닷넷 프레임워크와 함께 선택적으로 사용하고자 하는 need 가 생겨나기 시작했다. iBatis.net 이나 spring.net, log4.net 등의 닷넷 버전의 오픈 소스 프레임워크가 나타나게 되었다. Microsoft 는 ORM 기반의 Entity Framework 를 릴리즈 했다. 여기에서는 프레임워크와 엔티티 프레임워크에 대해서 소개를 하려 한다.

<프레임워크의 개념 >

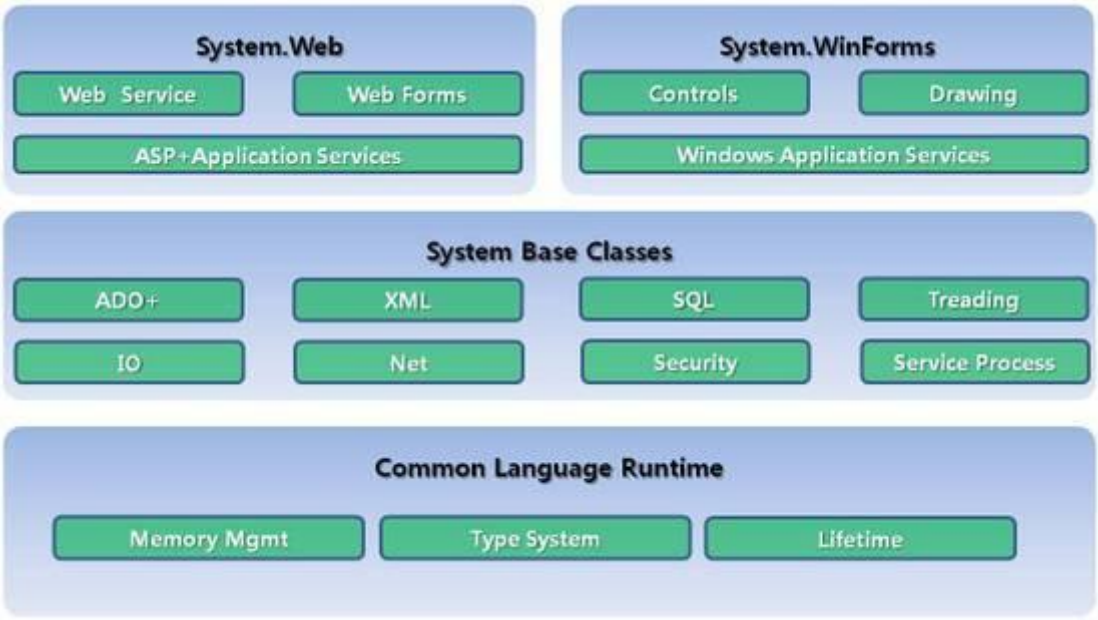
과연 프레임워크는 무엇일까? 지금 여러분들의 머릿속에 프레임워크에 관련된 뭔가가 두리뭉실하게 그려질 것이다. 프레임워크란 기반이 되는 뼈대라고 생각하면 된다. 예를 들어 건물을 짓는 것을 봤다면 건물이 다 지어지기 전까지 그 건물이 상가인지 아파트인지 한옥인지 알 수 없다. 다만 건물을 짓기 위한 공통된 작업 즉 초기에 땅을 파고 땅에서 물과 돌을 제거하고 기반을 단단히 다지는 작업을 하게 되는데 이런 밑그림, 기반 틀이 프레임워크라고 할 수 있다.

닷넷 개발자라면 닷넷 프레임워크 기반에서 개발을 수행하게 된다. 닷넷 프레임워크는 보시다시피 닷넷 Base Class Library 기반으로 응용프로그램을 구현 한다.



[그림 1-1] .NET Framework

사실 닷넷을 사용하는데 가장 큰 장점 중 하나가 Base Class Library 가 제공된다는 것인데 파일을 핸들링 하기 위해서 닷넷에서 제공되는 System.IO 를 DataBase 를 사용하기 위해 ADO.NET 을 이용하면 보다 빠르고 쉽게 응용프로그램을 구현할 수 있다. 이런 개념만 이해한다면 초보개발자도 닷넷 기술을 이용하여 응용프로그램을 빠르게 개발할 수 있다.



[그림 1-2] .NET Framework Class Library

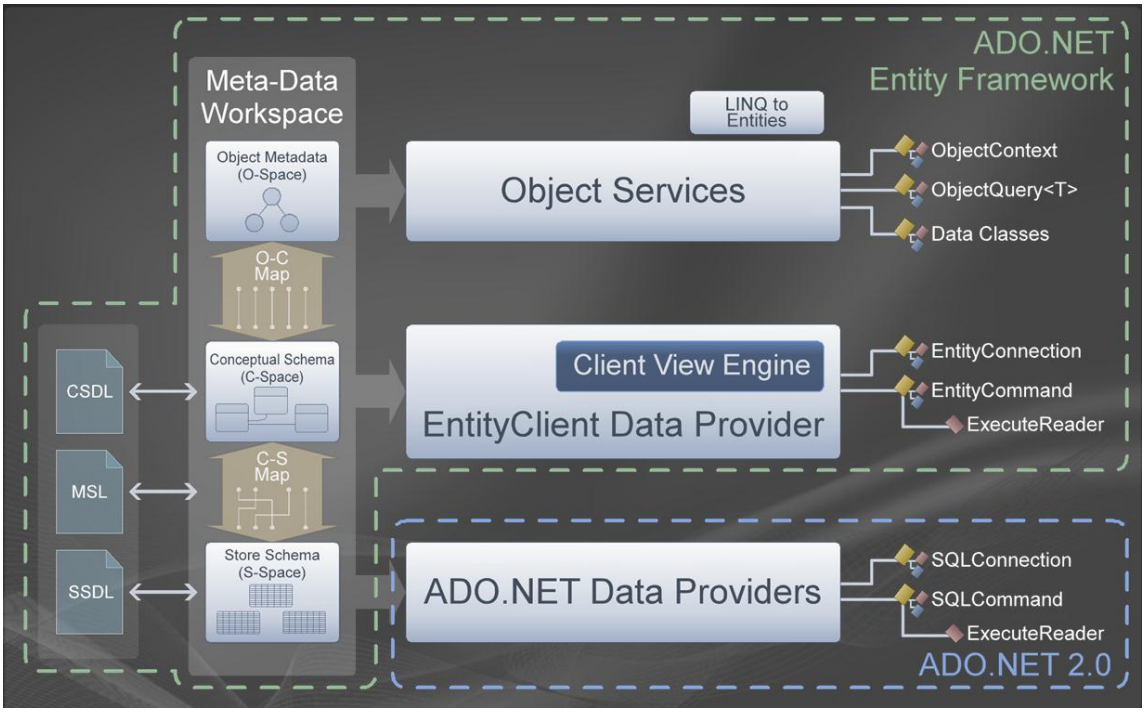
개발자 측면에서 프레임워크란 이처럼 응용프로그램을 만들기 위해서 기반이 되어주는 기능별로 모듈화가 되어 있고 재사용 가능한 구조, 틀이라고 정의할 수 있다.

<닷넷계 연계 가능한 오픈소스>

자바 진영에서는 수많은 프레임워크가 있다. 프로젝트 특성에 맞춰 프레임워크가 선택적으로 사용되기도 하죠. 웹 어플리케이션 영역에서 활용되는 벨로시티 프레임워크나 스프링 프레임워크, Data 매핑의 Hibernate, iBatis 프레임워크, 로깅관련 된 Log4J 프레임워크 등등의 프레임워크가 있다. 이 모든 프레임워크가 자바 진영에서는 활용될 수 있는 것은 그만큼 오프소스에 대해 열려있었기 때문이다. 최근에 들어서는 닷넷계에서도 닷넷 기반에서 오픈소스 프레임워크 접목하려는 시도가 점차 늘어가고 있다. ORM(Object Relational Mapping) 의 iBatis.NET 이나 NHibernate, Ioc(Inversion of Control)기반의 Spring.NET 등이 있다. 이는 닷넷 버전으로 자바에서 사용되었던 프레임워크가 컨버팅 된 닷넷 버전 프레임워크이다.

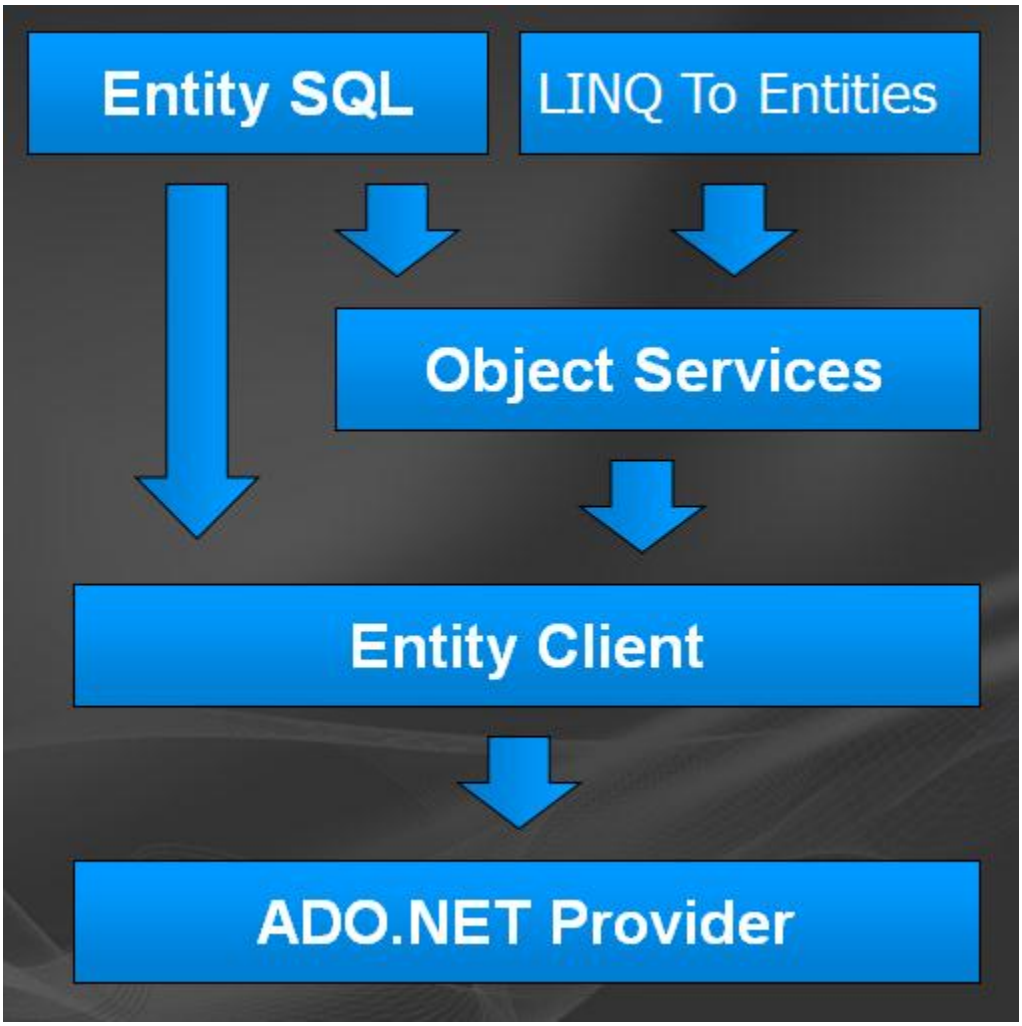
<위대한 탄생? 엔티티 프레임워크 탄생!>

오픈 소스에서 ORM 에 관련된 프레임워크가 인기를 끌게 되면서 MS 는 기존의 ADO.NET 에서 ORM 기반의 프레임워크를 Visual studio 2008 sp1 에 정식 릴리즈 했다. 그것이 바로 Entity Framework 이다.



[그림 1-3] Entity Framework 아키텍처

Entity Framework 는 Data 를 Entity 로 매핑 할 수 있는 것으로 우리가 흔히 쓰는 Database 개체들을 Object 화 할 수 있다. Entity Framework 가 릴리즈 될 당시에 .NET Framework 3.5 에서는 LINQ 가 포함되어 관계형 데이터를 보다 용이하게 질의할 수 있게 제공한다. Entity Framework 가 추가되면서 개발자로 하여금 더 높은 생산성의 향상을 기대하게 되었다. 또한 Entity Framework 는 직관적인 DAO(Data Access Object) 패턴의 응용프로그램을 구현하게 된다.



[그림 1-4] Service Stack

위 그림에서 보여주는 것과 같이 Entity Client 는 중간에서 기존의 Linq To Entities 와 Entity SQL 을 ADO.NET 모델로 확장을 한다.

[Entity Framework 강좌] 02. Entity Framework 4.0 기능 살펴보기

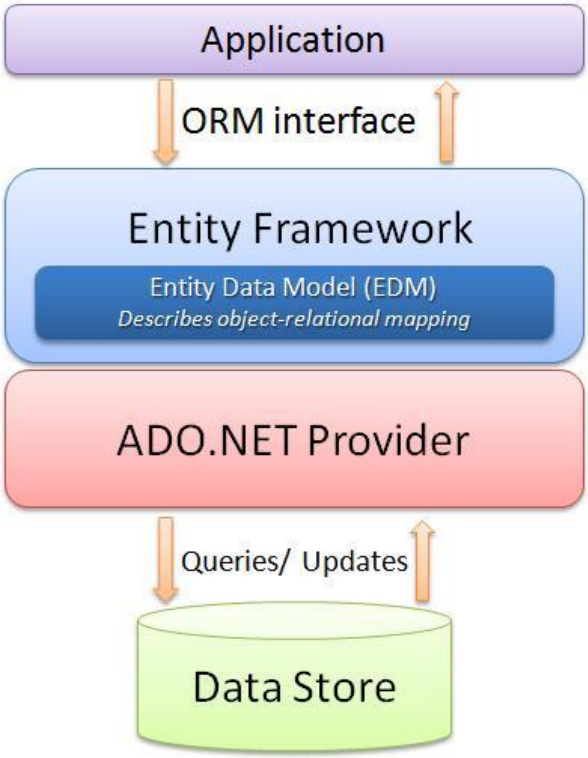
Entity Framework 4.0 기능 살펴보기

Entity Framework 는 데이터와 객체(Entity)관계를 쉽게 맺어줌으로 실제적으로 ORM 프레임워크이다.

<대표 기능>

1. 다양한 데이터베이스 서버 지원

Data Store 가 다음과 같으면 언제 어디서든 Entity Framework 를 사용 할 수 있다. MS SQL, Oracle, MySql, PostgreSQL, SQL Anywhere, DB2, Informix, U2, Ingres, Progress, Firebird, Synergy, Virtuoso 등이 있다. 저희가 주로 쓰는 MS SQL, Oracle, MySql 가 있는데 Entity Framework 는 ADO.NET Data Provider 상위에서 빌드되기 때문에 이런 많은 데이터베이스 지원이 가능하다.



[그림 2-1] Entity Framework

1. 통합된 Visual Studio 도구 제공

Visual Studio Tool 에서 시각적인 Entity 모델과 기존 데이터베이스 개체 모델화를 자동으로 생성해준다. 또한 새로운 데이터베이스도 Visual Studio 에서 모든 권한을 편집하여 배포 가능하다. Visual Studio 의 가장 큰 장점인 드래그 앤 그롭이 가능하다는 것이다.

2. POCO(Plain Old CLR Objects) 지원

기존 버전에서는 데이터 저장소와 논리적 객체(Entity)간의 연관성이 표현되지 않았다. 이를 지속성 무시 개체 POCO 라고 불리는데 이번 버전에서는 영속성을 지원하는 POCO Entity 가 제공된다. POCO Entity 는 System.Data.Objects.ObjectContext 클래스를 상속 받는데 이 클래스에서 제공되는 메서드는 데이터 저장소와 엔티티간의 영속성을 지원 가능하게 해준다.
http://msdn.microsoft.com/ko-kr/library/system.data.objects.objectcontext.aspx

3. Model-First 지원

Entity Framework 이전 버전에서 개념적 모델을 만들 수 있다. 다만 개념적 모델은 데이터베이스 마법사를 통해서 생성되기 때문에 모델은 DB 기반의 모델에 존속 될 수 밖에 없었다. Entity Framework 4.0 에서는 기존방식도 지원하고 개념적 모델이 DB 에 물리적 개체가 없어도 모델링이 가능하게 지원한다. 이를 Model-First 라고 한다.

4. 관계 Object 의 지연 실행

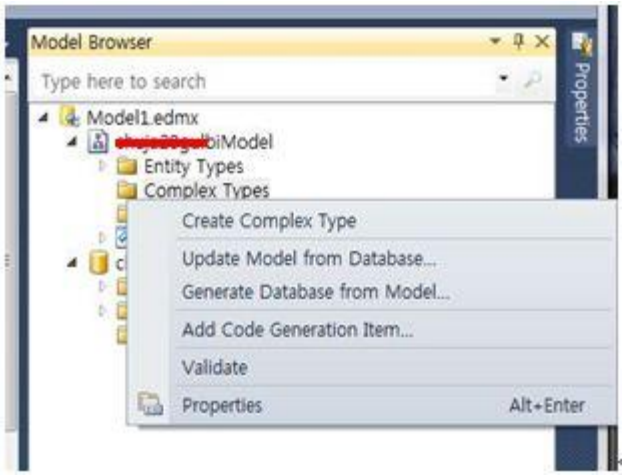
지연 실행에 대해서는 기존 버전에도 존재하며 LINQ 의 대표 기능이기도 하다. 다만 관계형에 대해서는 표현이 되지 않았다. 이번 버전에서 쿼리 결과가 명시적으로 탐색 속성에서 관계형 개체를 확인 할 수 있다.

5. 엔티티 질의 LINQ 함수

Entity Framework 초기 버전에서는 함수의 지원이 제한되어 있었다. 여기서 말하는 함수는 저장 프로시져나 데이터베이스 UDF 를 말한다. 새로운 EntityFunctions 과 SqlFunctions 클래스가 이 이슈를 해결하기 위해 추가 되었다. 이 클래스들은 개발자에게 LINQ 로 엔티티 질의를 할 수 있게 기능을 제공한다.

6. Complex Type 지원

Visual Studio 2010 Entity Data Model Designer 에서 원하는 복잡한 타입을 쉽게 정의 할 수 있다. 모델 탐색기에서 트리로 보여준다. 아래 그림처럼 모델 탐색기에서 Complex Type 을 생성 할 수 있으며 여기서 정의된 타입은 Entity 속성으로 사용된다.



[그림 2-2] Model Browser

7. 모델 브라우저 개선

Entity Framework 4.0 이 릴리즈되면서 모델 브라우저에 몇 가지들이 개선되었다.

- 기본 데이터베이스가 변경 사항이 있을 때 모델 업데이트
- 모델에서 객체 삭제하기
- 스토리지와 개념적 모델에서 지정한 문자열 검색 기능
- 디자인 화면에서 매핑 된 Entity 타입 찾기

<Entity Framework 의 장점>

1. 개발시간 절감된다.
2. 개발자는 응용프로그램을 구현하는 입장에서 객체 모델의 관점에서 작업할 수 있다.
3. 응용프로그램은 독립적인 개념적 모델을 지원함으로써 실제 데이터 저장소에 대한 종속이 해방될 수 있다.
4. 객체 모델과 특정 데이터 저장소의 스키마 사이의 매핑인 응용프로그램의 코드를 변경하지 않고도 가능하다.
5. LINQ 지원으로 개념적 모델에 대한 질의 시에 인텔리센스가 제공되며 컴파일 타임에서 구문 유효성 검사가 제공된다.

참고 URL

http://msdn.microsoft.com/en-us/data/dd363565.aspx
http://msdn.microsoft.com/en-us/data/aa937709
http://msdn.microsoft.com/en-us/data/aa937723
http://archive.msdn.microsoft.com/cs2010samples

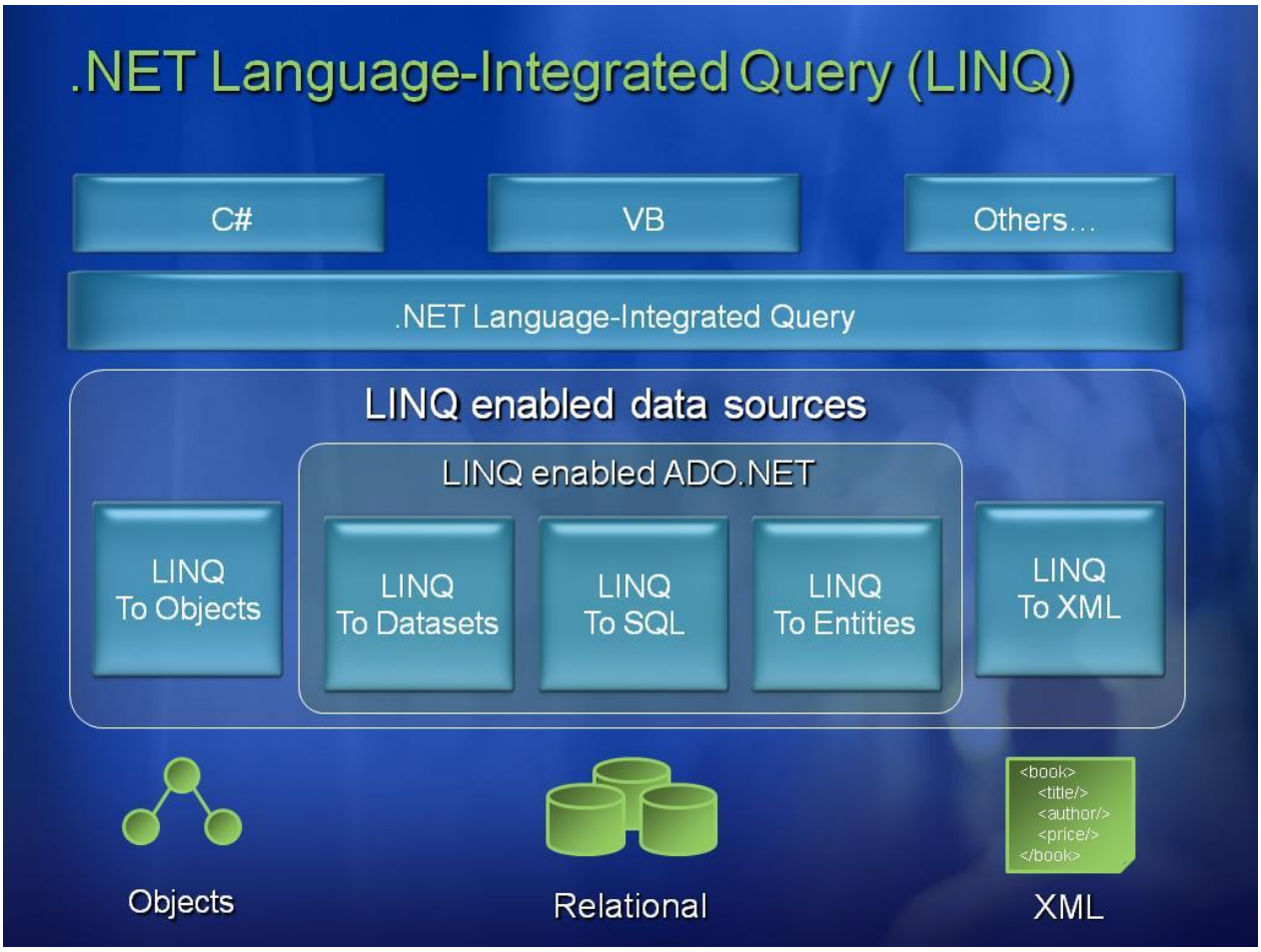
[Entity Framework 강좌] 03. LINQ 알고 가자

LINQ 알고 가자

엔티티 프레임워크에서는 엔티티를 질의하는데 LINQ 문을 사용한다. 하여 이번에는 LINQ 의 기본적인 사용법에 대해서 알아보자.

<LINQ 란?>

LINQ(Language Integrated Query)는 통합언어 쿼리로 .NET Framework 3.5 에 포함되었다. LINQ to Object, LINQ to SQL, LINQ to XML 로 크게 나눌 수 있는데 이는 데이터베이스, XML 등등 개체화하는 데이터 소스에 대해서 전반적으로 쉽게 질의 할 수 있게 제공한다.



[그림 3-1] .NET LINQ

이번 세션의 목표는 LINQ 문법을 익히는데 중점을 두며 아래 URL 에 가면 LINQ 샘플을 다운로드 받아 볼 수 있는데 샘플을 기준으로 각각의 표현식을 설명하려 한다. 부족한 부분은 샘플을 다운받아서 한번씩 살펴보시기 바란다.
LINQ 샘플 다운로드 URL - http://msdn.microsoft.com/ko-kr/vcsharp/aa336746.aspx

LINQ 는 쿼리 표현식과 람다식 함수로 나눠 작성할 수 있다. 예제는 주로 쿼리 표현식으로 작성하겠다. 더 람다식에 대해 더 자세히 공부하고 싶으신 분은 아래 링크를 참고할 수 있다.
람다 식(C# 프로그래밍 가이드)- http://msdn.microsoft.com/ko-kr/library/bb397687.aspx

<기본 표현식>

Select

기본 데이터소스에서 특정 개체를 조회해오는 구문이다. 구문 작성은 SQL 구문과 비슷해서 쉽게 익힐 수 있다.

```
    public static void Select1()
    {
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

        var numsPlusOne = from n in numbers
                          select n + 1;

        Console.WriteLine("Numbers + 1:");
        foreach (var i in numsPlusOne)
        {
            Console.WriteLine(i);
        }
    }
}
```

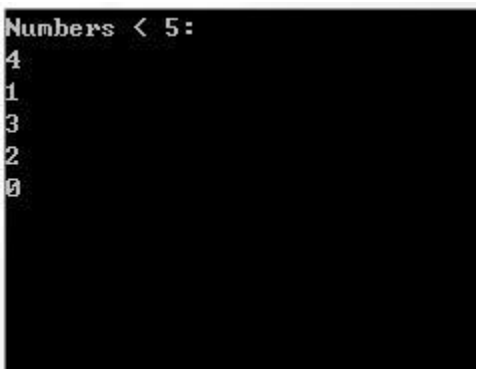
```
Numbers + 1:
6
5
2
4
10
9
7
8
3
1
```

[그림 3-2] Select 실행 결과

Where

해당 조건에 맞는 데이터를 조회할 경우 사용한다.

```
public static void Where1()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var lowNums = from n in numbers
                  where n < 5
                  select n;

    Console.WriteLine("Numbers < 5:");
    foreach (var x in lowNums)
    {
        Console.WriteLine(x);
    }
}
```



[그림 3-3] Where 실행 결과

LINQ 는 지연된 쿼리라는 특성을 가지고 있다. 이는 데이터 조건이 실행되어 결과 집합으로 가지고 있는것이 아니라 조건을 출력하는 시점에서 실행이 되는 것을 뜻한다. 즉 위의 표현식에서 살펴보면 실제로 위에 해당되는 쿼리가 실행되는 시점은 출력을 요청하는 foreach 구문 내부다. 이를 지연된 쿼리라고 불리며 개발자의 설정에 따라 즉시실행을 할 수도 있다.

OrderBy

데이터 정렬하는 표현식이다.

```
public static void OrderBy1()
{
    string[] words = { "cherry", "apple", "blueberry" };

    var sortedWords = from w in words
                      orderby w
                      select w;

    Console.WriteLine("The sorted list of words:");
    foreach (var w in sortedWords)
    {
        Console.WriteLine(w);
    }
}
```

```
The sorted list of words:
apple
blueberry
cherry
```

[그림 3-4] OrderBy 실행 결과

역정렬을 원하는 경우 "descending" 구문만 추가해주면 된다.

```
var sortedDoubles = from d in doubles
                    orderby d descending
                    select d;
```

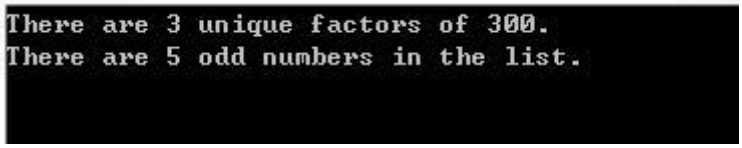
<집계 표현식>

LINQ 는 집계 연산을 수행하는 함수를 기본적으로 제공한다. 기본함수에는 Count, Sum, Min/Max, Average, Aggregate 등등이 있다. 대표해서 몇 가지만 알아보자.

Count

조회된 데이터의 Row 수를 리턴한다. LINQ 는 기본적인 메서드를 제공하는데 Count 메서드만 호출하면 조회된 결과의 개수를 반환한다. Count2 메서드는 조건에 해당하는 결과만 반환하게 된다.

```
public void Count1()
{
    int[] factorsOf300 = { 2, 2, 3, 5, 5 };
    int uniqueFactors = factorsOf300.Distinct().Count();
    Console.WriteLine("There are {0} unique factors of 300.", uniqueFactors);
}

public void Count2()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    int oddNumbers = numbers.Count(n => n % 2 == 1);
    Console.WriteLine("There are {0} odd numbers in the list.", oddNumbers);
}
```



[그림 3-5] Count 실행 결과

Sum

Sum 은 집합의 합계를 반환한다. Count 와 동일하게 Sum 메서드만 호출하면 된다.

```
public void Sum1()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    double numSum = numbers.Sum();
    Console.WriteLine("The sum of the numbers is {0}.", numSum);
}

public void Sum2()
{
    string[] words = { "cherry", "apple", "blueberry" };
    double totalChars = words.Sum(w => w.Length);
    Console.WriteLine("There are a total of {0} characters in these words.", totalChars);
}
```

```
The sum of the numbers is 45.
There are a total of 20 characters in these words.
```

[그림 3-6] Sum 실행 결과

Min/Max

조회된 결과값의 최소값/최대값을 반환한다.

```
public static void Min1()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    int minNum = numbers.Min();
    Console.WriteLine("The minimum number is {0}.", minNum);
}

public static void Max1()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    int maxNum = numbers.Max();
    Console.WriteLine("The maximum number is {0}.", maxNum);
}
```

```
The minimum number is 0.
The maximum number is 9.
```

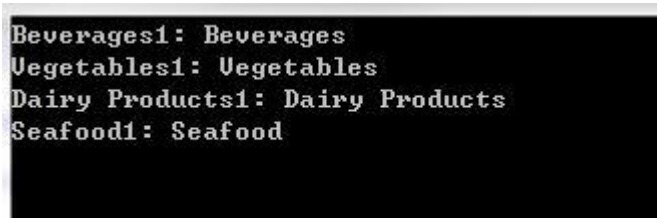
[그림 3-7] Min/Max 실행 결과

<기타 표현식>

Join

서로 다른 데이터를 조인하여 결과를 반환하는 구문이다. 관계형 데이터베이스 경우 이런 구문을 자주 사용하여 데이터를 조회한다. 아래 소스는 카테고리가 동일한 데이터만 출력하는 예제이다.

```
public static void Join1()
{
    string[] categories = new string[]{ "Beverages", "Condiments", "Vegetables", "Dairy Products", "Seafood" };
    List<Product> products = GetProductList();

    var q = from c in categories
            join p in products on c equals p.Category
            select new { Category = c, p.ProductName };

    foreach (var v in q)
        Console.WriteLine(v.ProductName + ": " + v.Category);
}

private static List<Product> GetProductList()
{
    List<Product> products = new List<Product>();
    products.Add(new Product("Beverages", "Beverages1"));
    products.Add(new Product("Vegetables", "Vegetables1"));
    products.Add(new Product("Dairy Products", "Dairy Products1"));
    products.Add(new Product("Seafood", "Seafood1"));
    products.Add(new Product("Source", "Source1"));

    return products;
}
```



[그림 3-8] Join 실행 결과

Into

Database 에 프로시져에서 임시로 데이터를 저장 해놓는 temp table 을 사용할 경우가 있다. 그와 같이 into 문도 임시로 데이터를 저장해야 할 경우에 사용하는 구문이다. 즉 임시 저장소를 만드는 것이지요~

```
var orderGroups = from prod in products
                  group prod by prod.Category into prodGroup
                  select prodGroup;
```

소스를 보면 prod 라는 개체에서 Category 별로 그룹을 지었다. 그 결과를 into 구문을 사용하여 prodGroup 에 담았다.

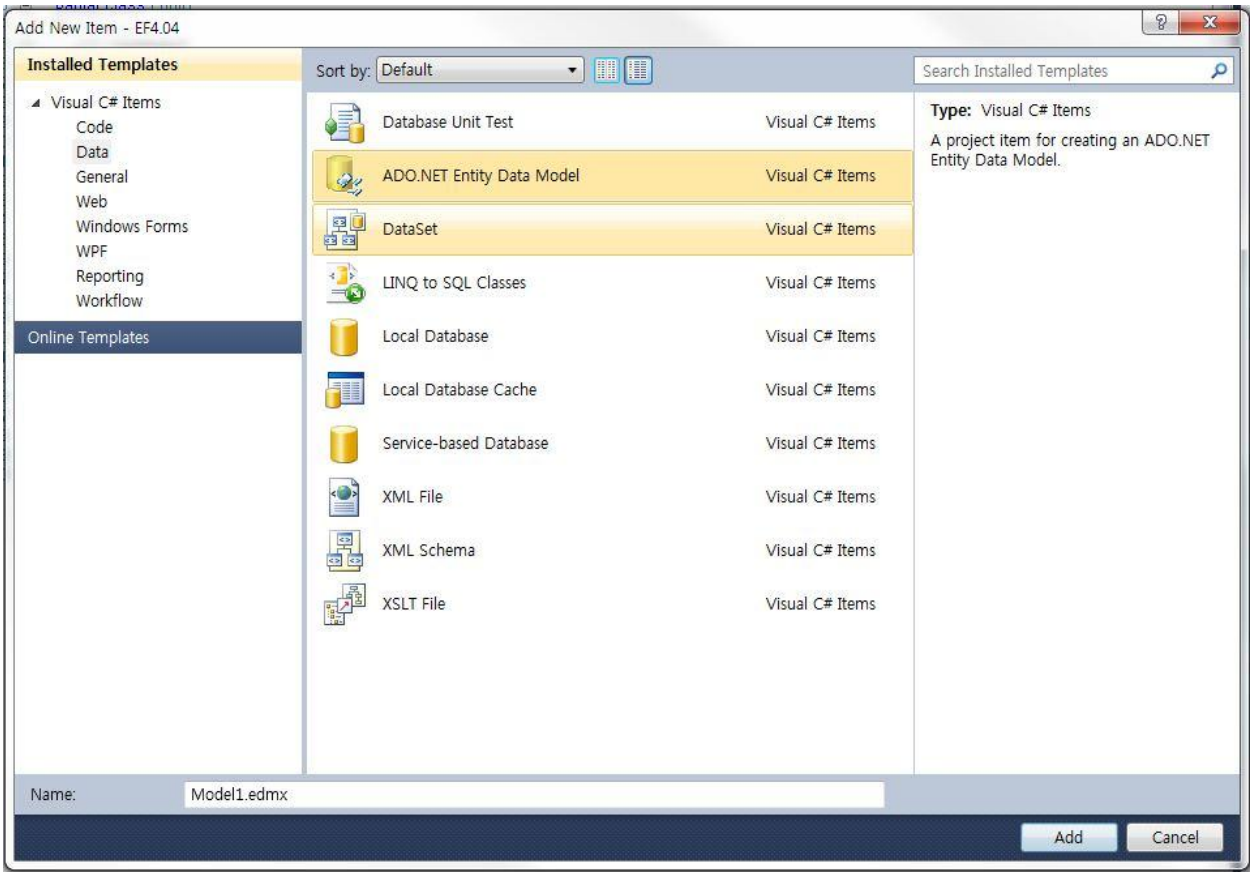
[Entity Framework 강좌] 04. Database-First VS Model-First

Database-First VS Model-First

이번에 알아 볼 내용은 EDM(Entity Data Model)이다. Entity Framework 4.0 기능에서 Model-First 지원이 있었다. 기존 버전과의 차이와 해당 기능에 대해서 자세히 살펴보는 시간이 되겠다.

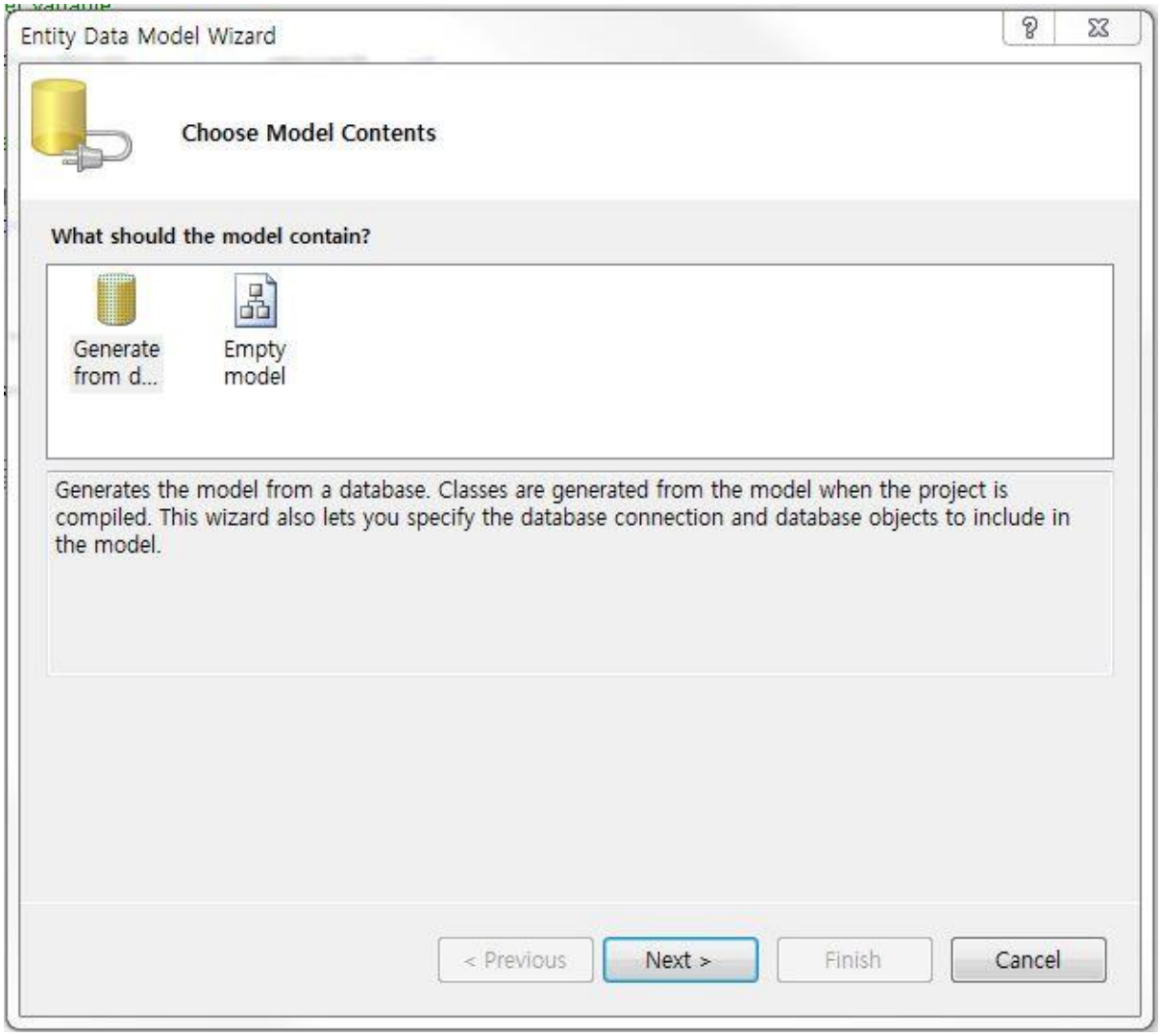
<Database-First>

이전 버전의 방식으로 데이터베이스 개체들을 엔티티화하는 것으로 실제로 존재하는 개체들 즉, Database 기반의 데이터 모델이라고 보시면 된다. Visual Studio 에서 간단한 프로젝트를 생성하시고 프로젝트에서 Item 항목을 추가 해보겠다. Item 에서 "ADO.NET Entity Data Model"을 선택해 준다.



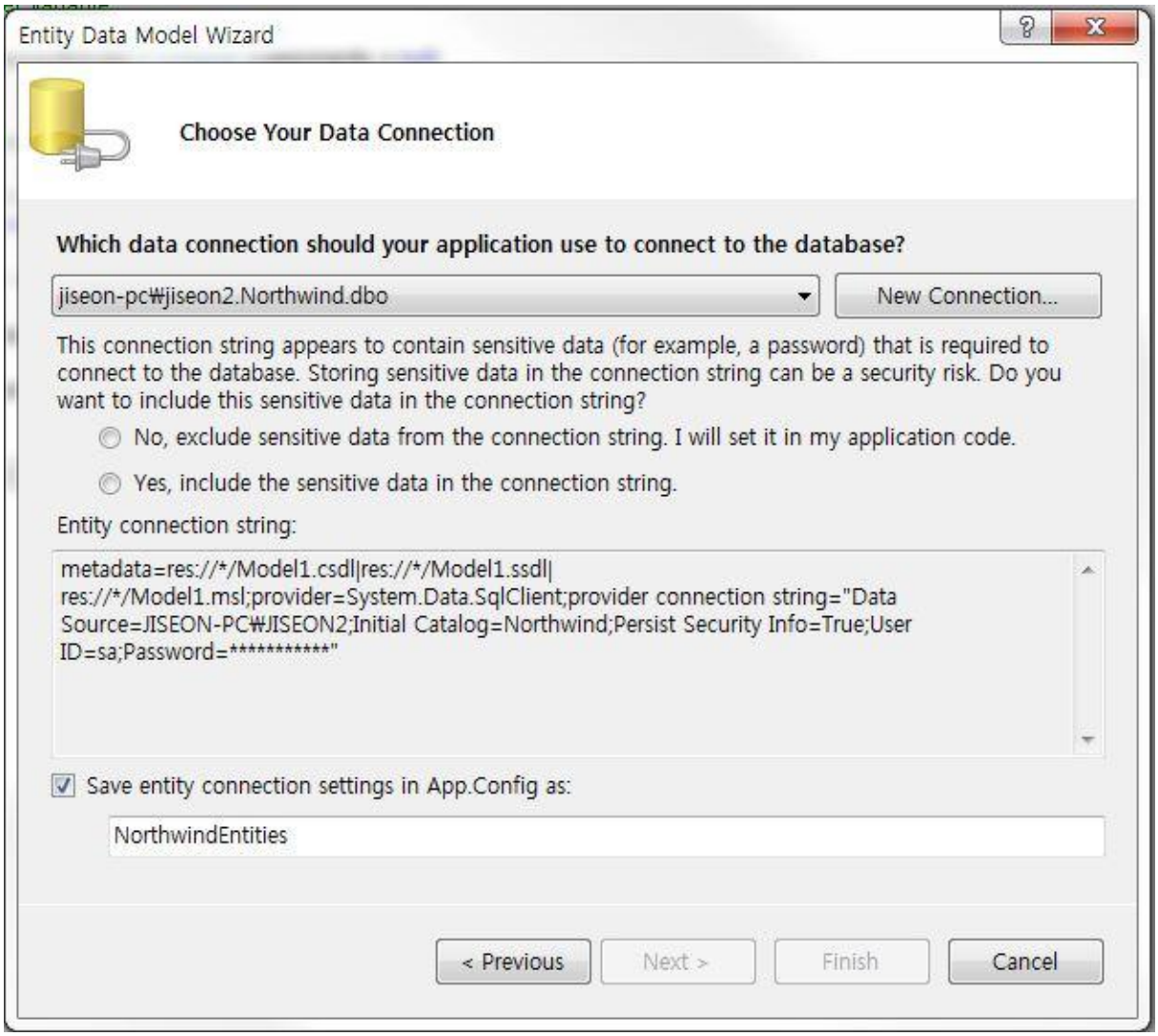
[그림 4-1] 항목 선택

추가하면 다음과 같은 화면이 나오는데 "Next"를 클릭한다.



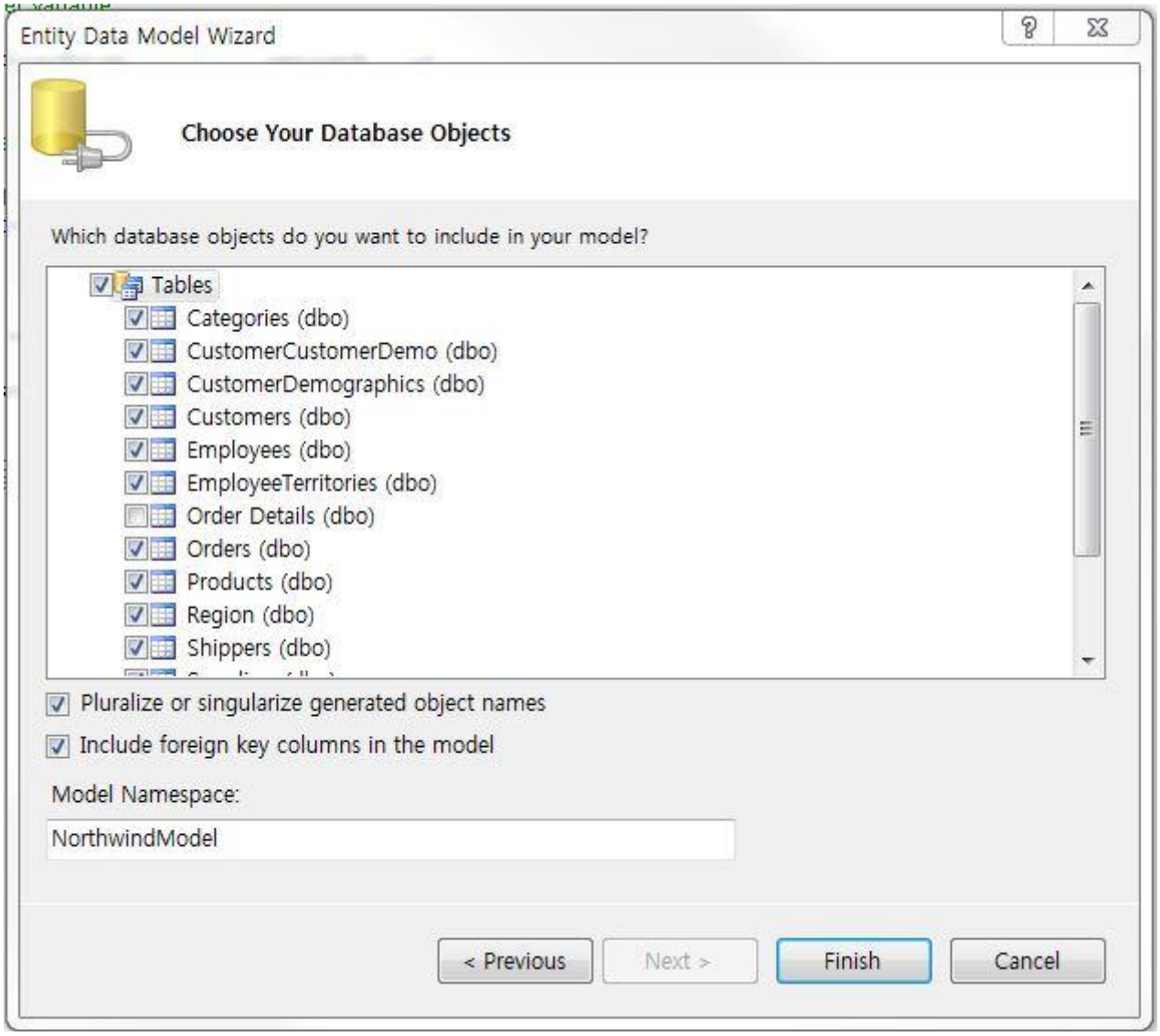
[그림 4-2] EDM 마법사

연결정보를 선택하고 App.Config 에 들어가는 연결명을 기입해준다. Default 값을 사용하셔도 된다. ConnectionString 설정 값이다.



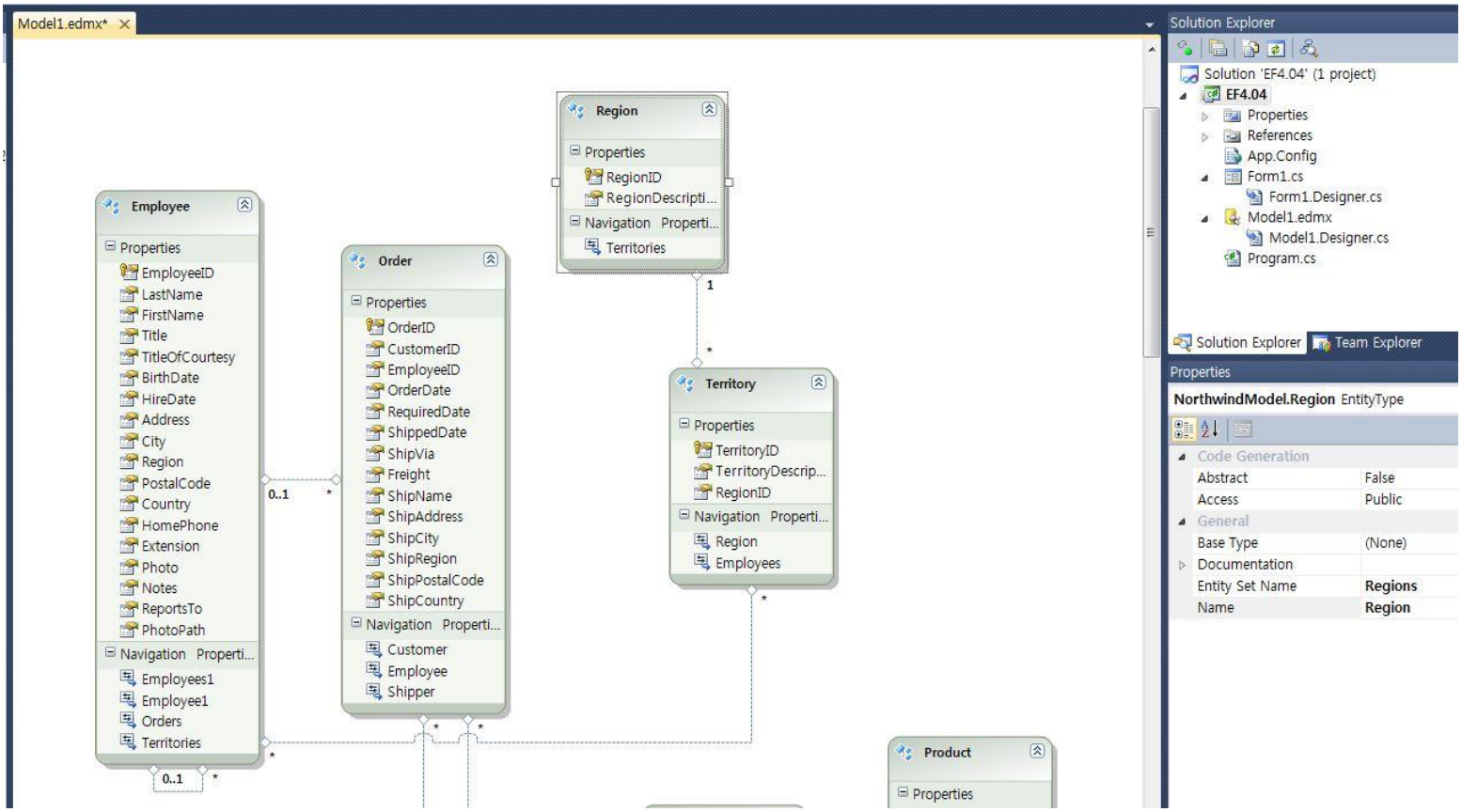
[그림 4-3] 데이터베이스 연결 설정

가져올 데이터베이스 개체들을 선택하고 모델 네임스페이스명을 기입해준다.



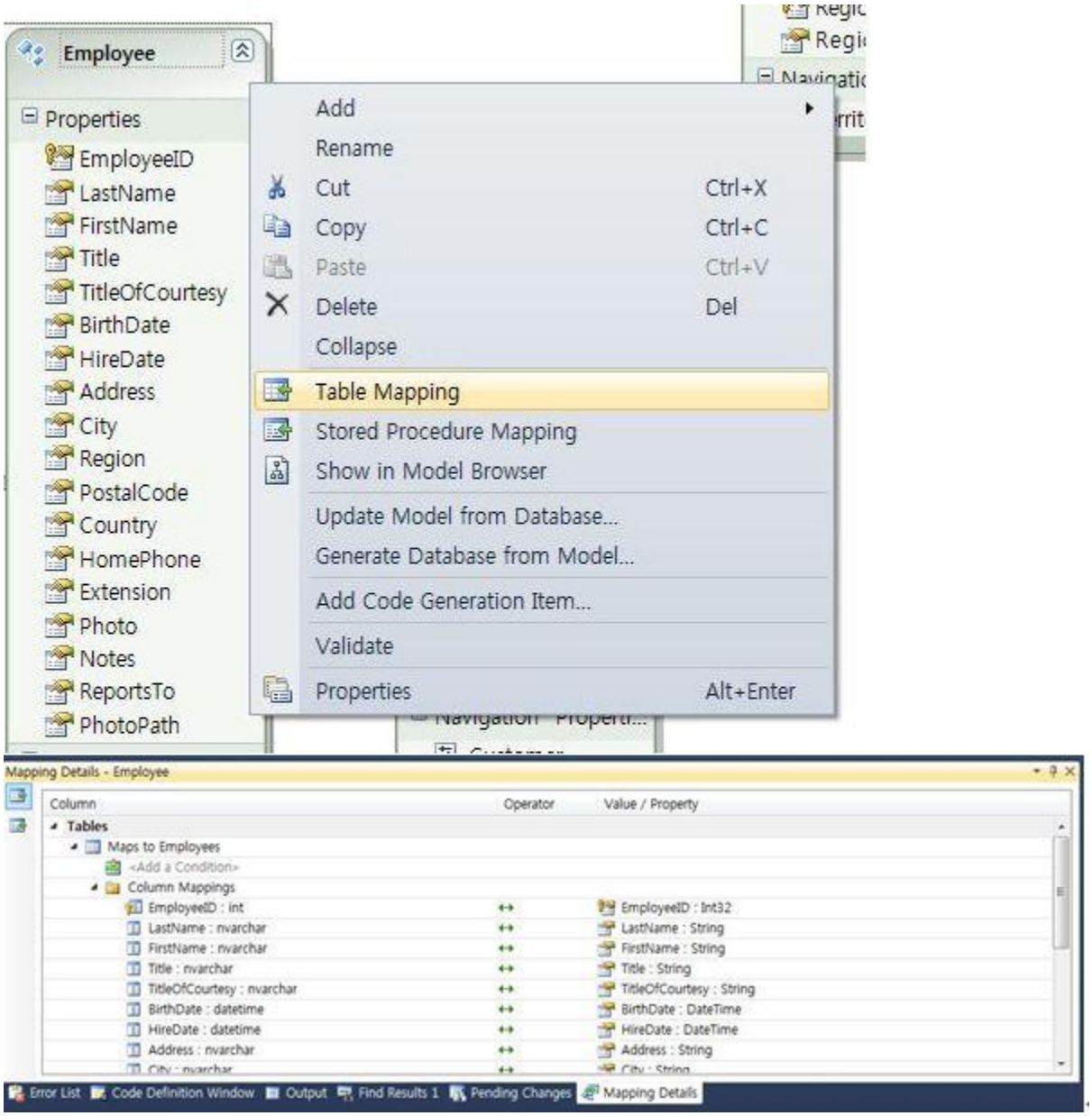
[그림 4-4] 개체 선택

보시다시피 데이터베이스 개체들 기준의 엔티티들이 자동으로 생성 되었다.



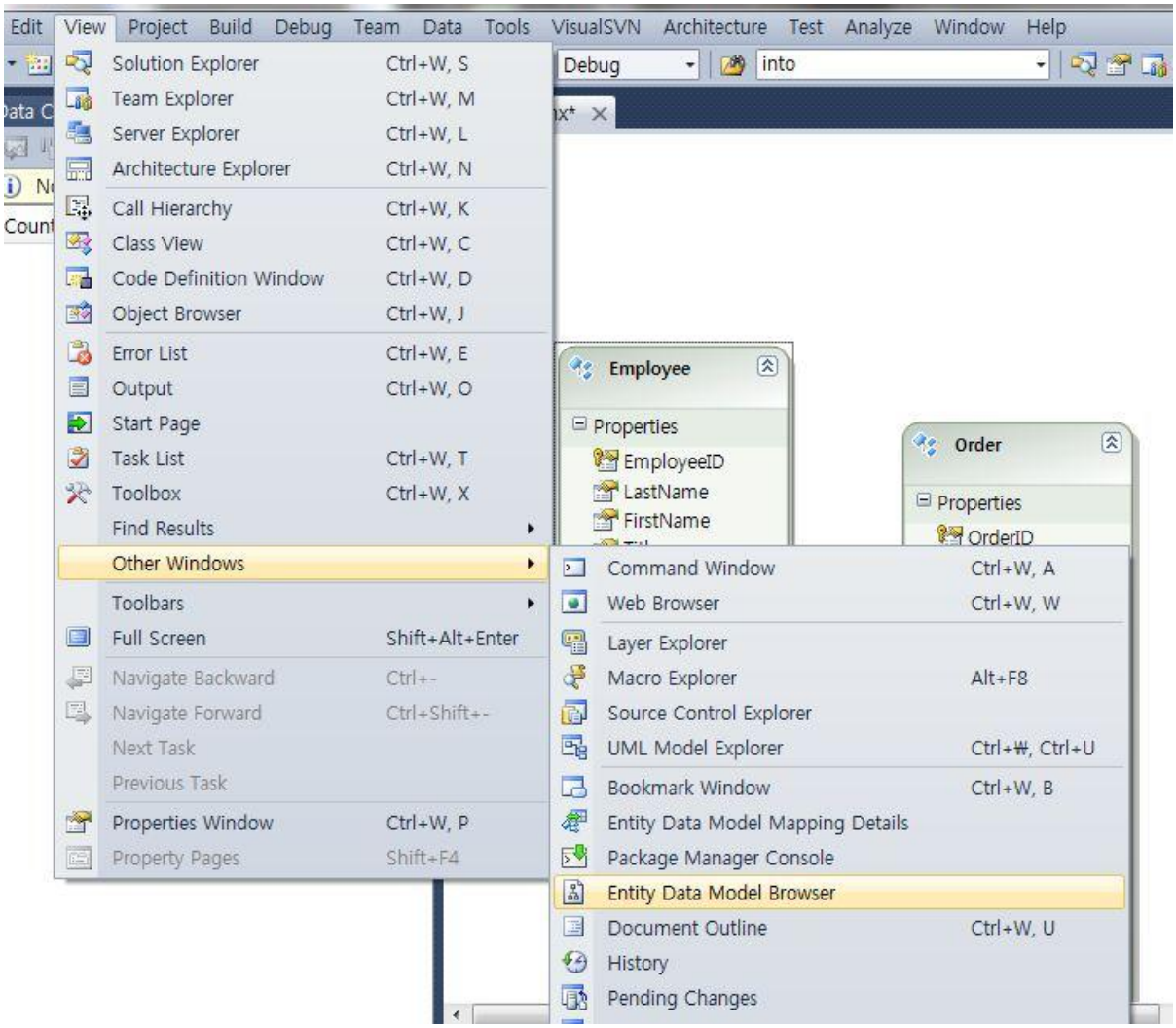
[그림 4-5] EDM

엔티티를 하나 선택하시고 마우스 오른쪽버튼을 클릭해서 "Table Mapping" 메뉴를 선택하시면 엔티티 상세정보가 출력된다. 엔티티와 실제 DB 개체관의 관계도 표현도 가능하다.



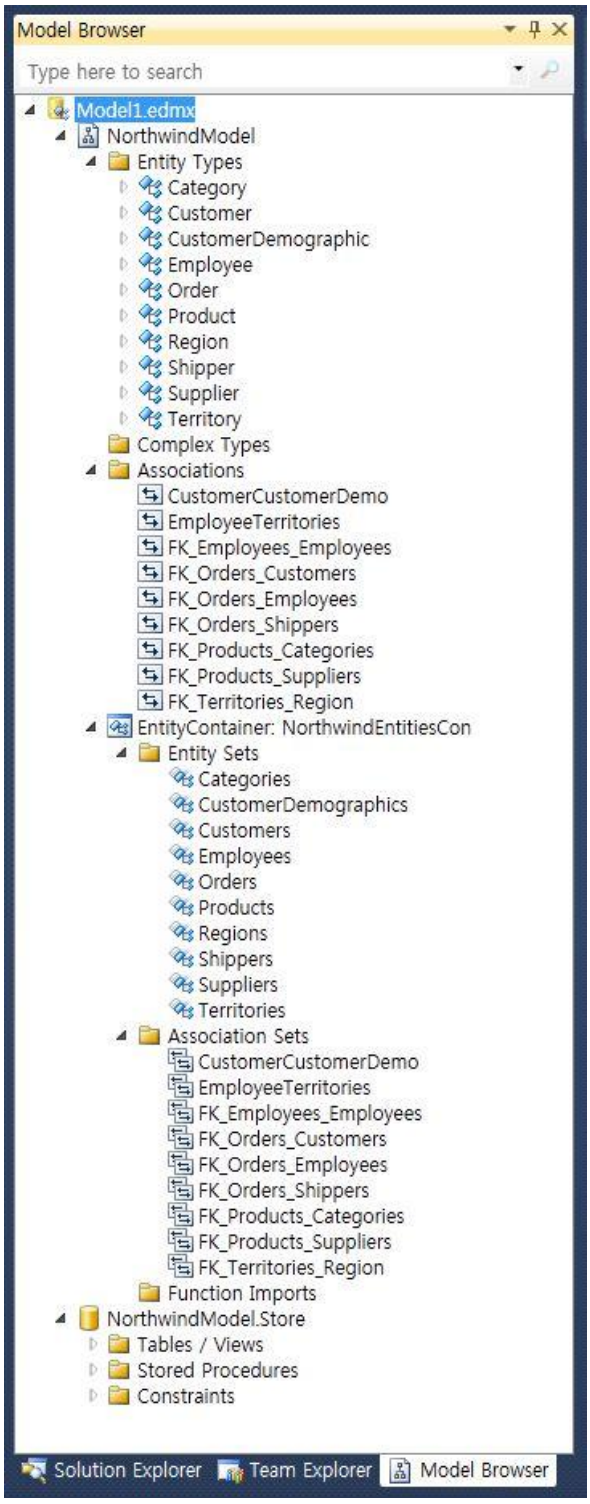
[그림 4-6] 테이블 매핑

추가로 View 메뉴에서 “Entity Data Model Browser”를 클릭하면 모델의 엔티티들을 볼 수 있다.



[그림 4-7] 뷰 > EDM 브라우저

엔티티는 모델에 정의된 객체들이다. 연계(Association)는 두 엔티티간의 관계이다. EDMX 의 객체들이 트리뷰로 출력된다.



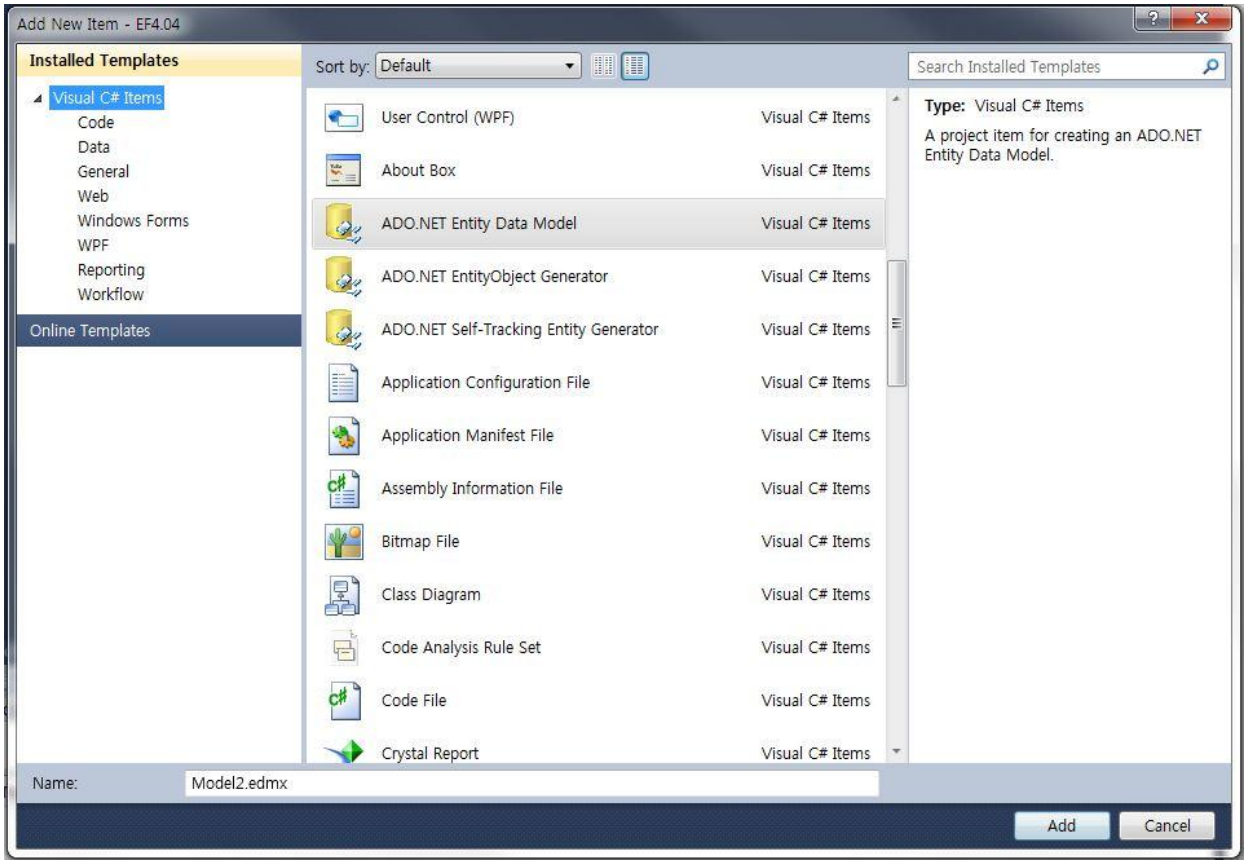
[그림 4-8] Entity Data Model Browser

지금까지 데이터베이스 기준의 모델을 생성하는 것을 확인하였다. 물리적 개체에서 논리적 모델로 표현되는 것이다. 이를 Database-First 라고 한다.

<Model-First>

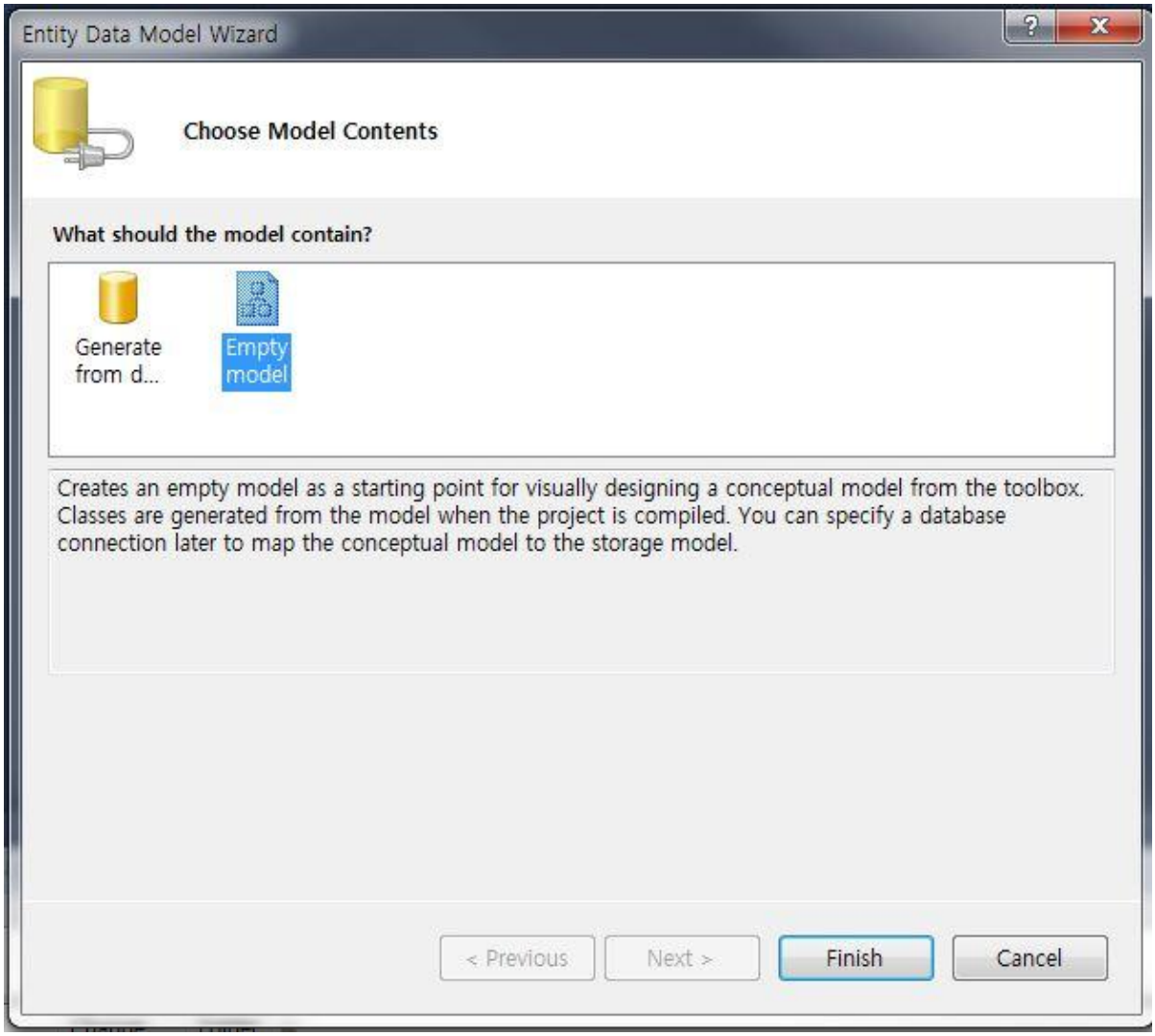
다음은 모델 우선이다. 데이터 모델링을 해보셨다면 아시겠지만 보통 **개념모델 ➔ 논리모델➔ 물리모델** 순으로 진행된다. 이전 버전의 엔티티 프레임워크는 어쩜 거꾸로였죠. ㅋ 엔티티 프레임워크 4.0 에서는 개념 모델링 후 저장소에 개체들을 생성할 수 있는 기능이 추가되었다. 또 시작해보죠!

"ADO.NET Entity Data Model" 항목을 추가해준다



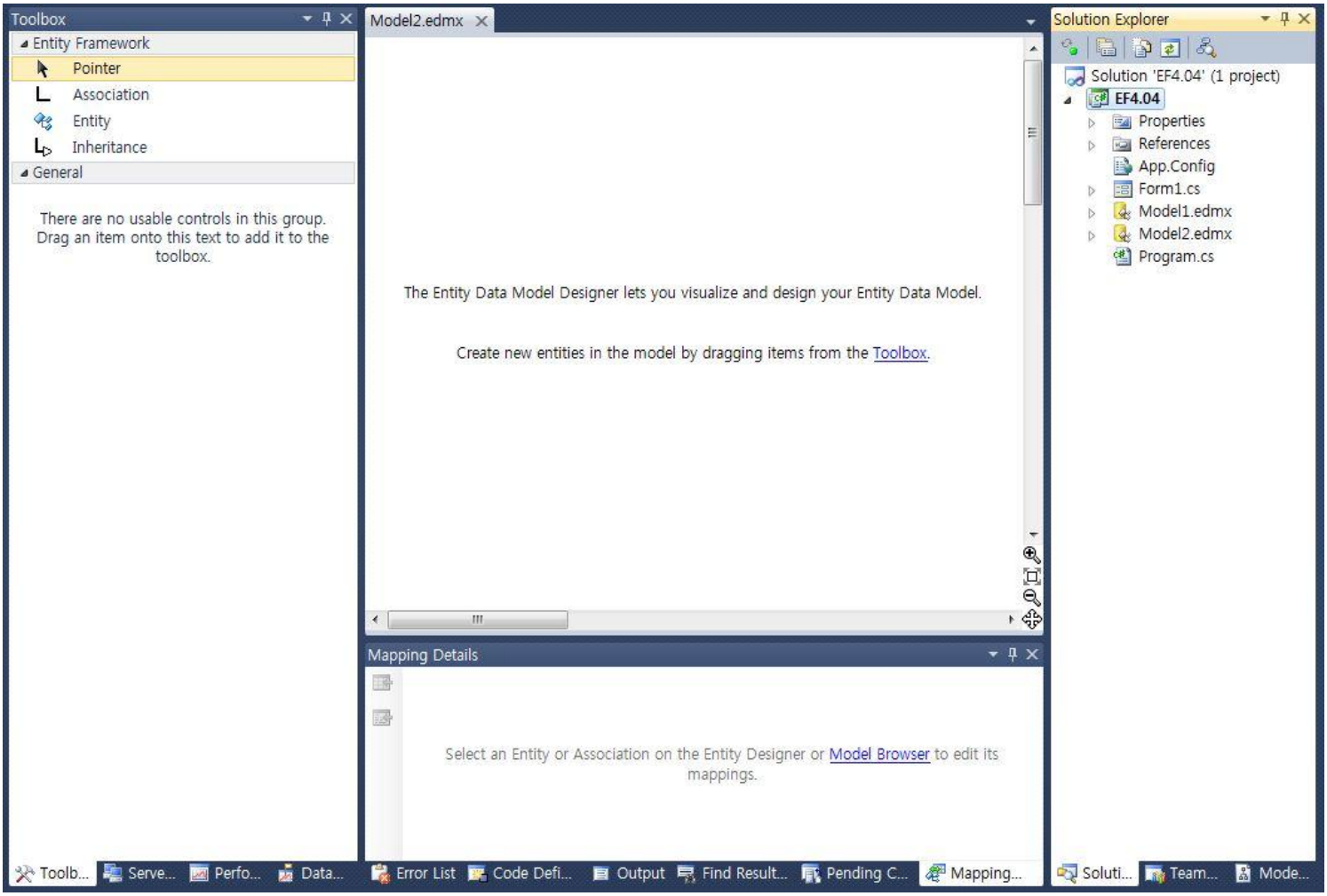
[그림 4-9] 항목 선택

빈 모델을 선택하고 완료버튼 클릭!



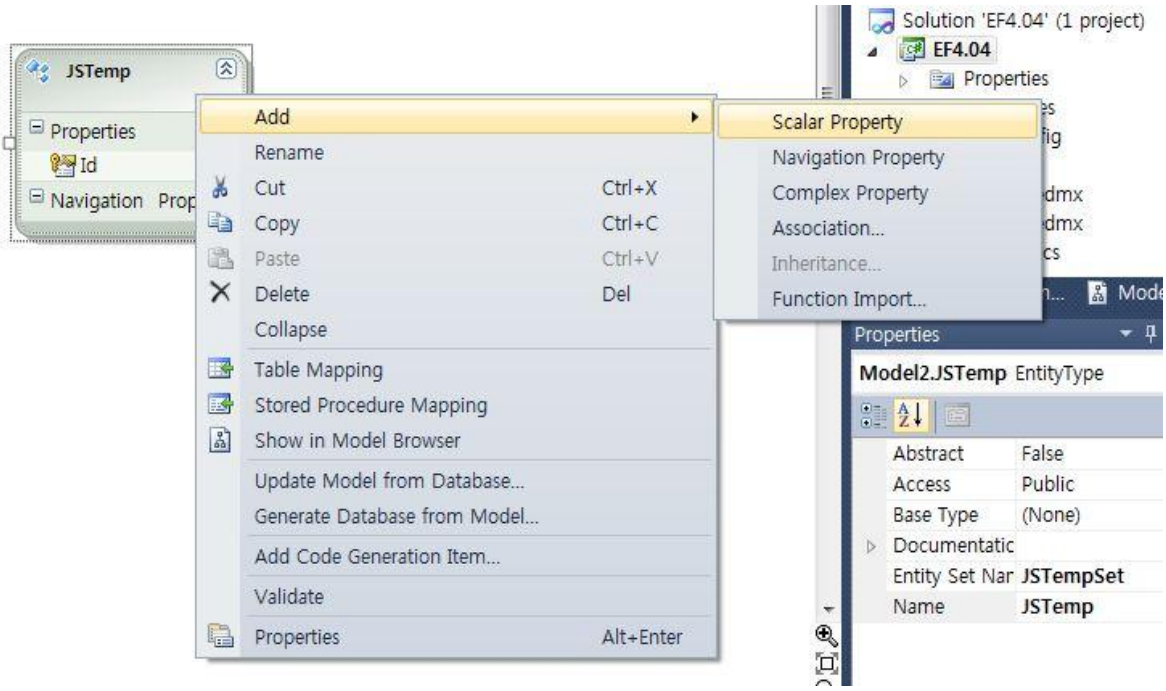
[그림 4-10] EDM 마법사

다음 화면처럼 빈화면이 나옵니다. Toolbar 를 보면 엔티티를 그릴 수 있는 도구들이 있다. 윗부분에서 언급했지만 한번 더! 엔티티는 모델에 정의된 객체들이고 연계(Association)는 두 엔티티간의 관계이다. 상속(Inheritance)은 두 엔티티간의 상속관계를 표현할 경우 사용된다.



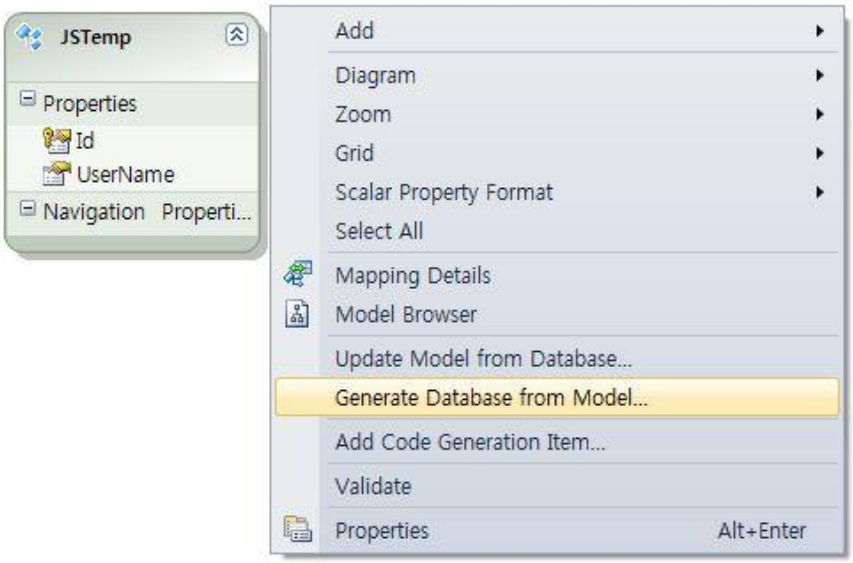
[그림 4-11] 빈 모델

도구들을 이용해서 엔티티 하나를 그려보겠다. 엔티티를 그리고 마우스 오른쪽을 클릭해서 속성 하나를 추가해보겠다.



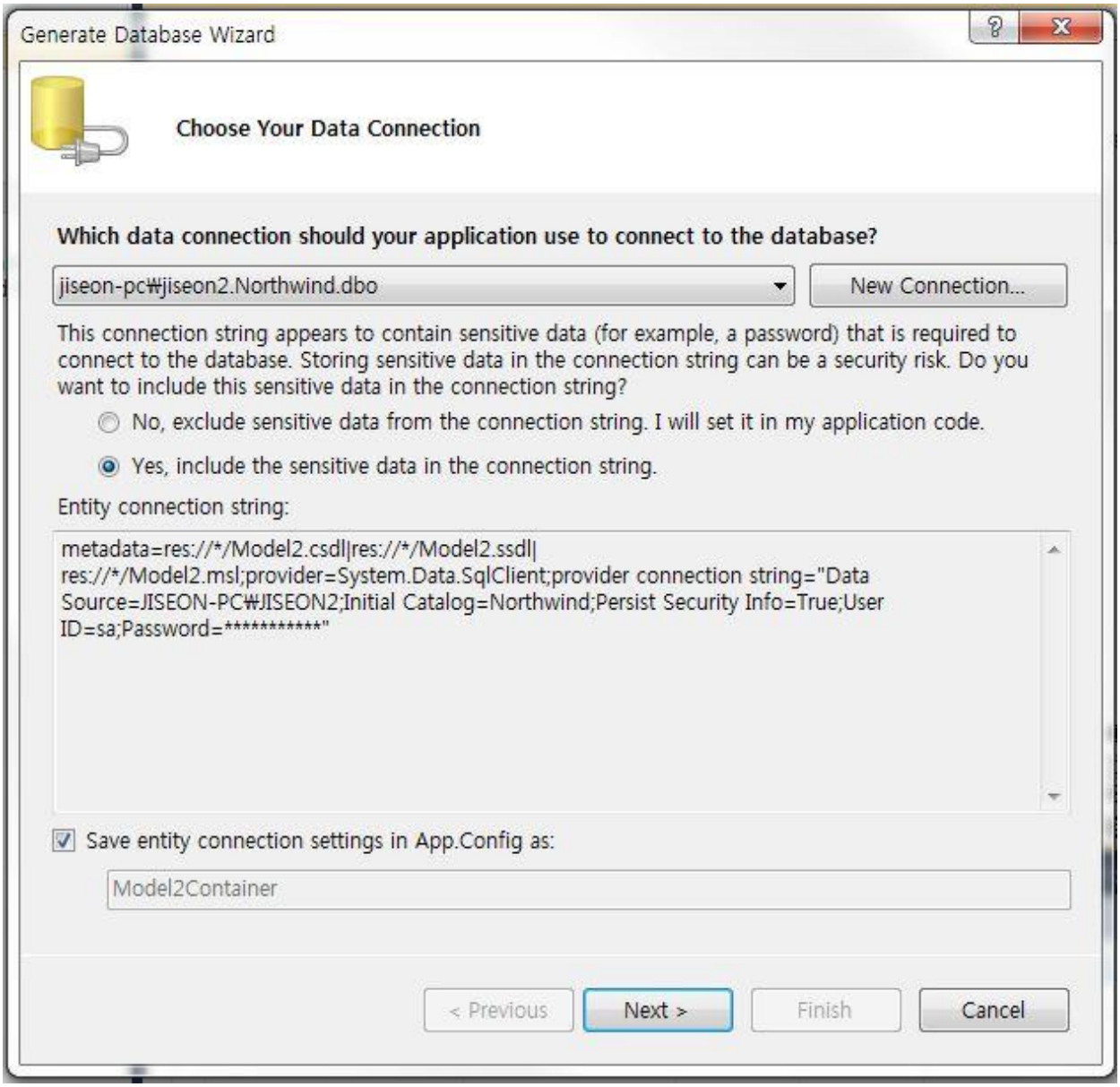
[그림 4-12] 엔티티 속성 추가

모델링 된 엔티티를 데이터 저장소에 생성해보겠다. 마찬가지로 마우스 오른쪽을 클릭해서 "Generate Database from Model..."을 선택하면...



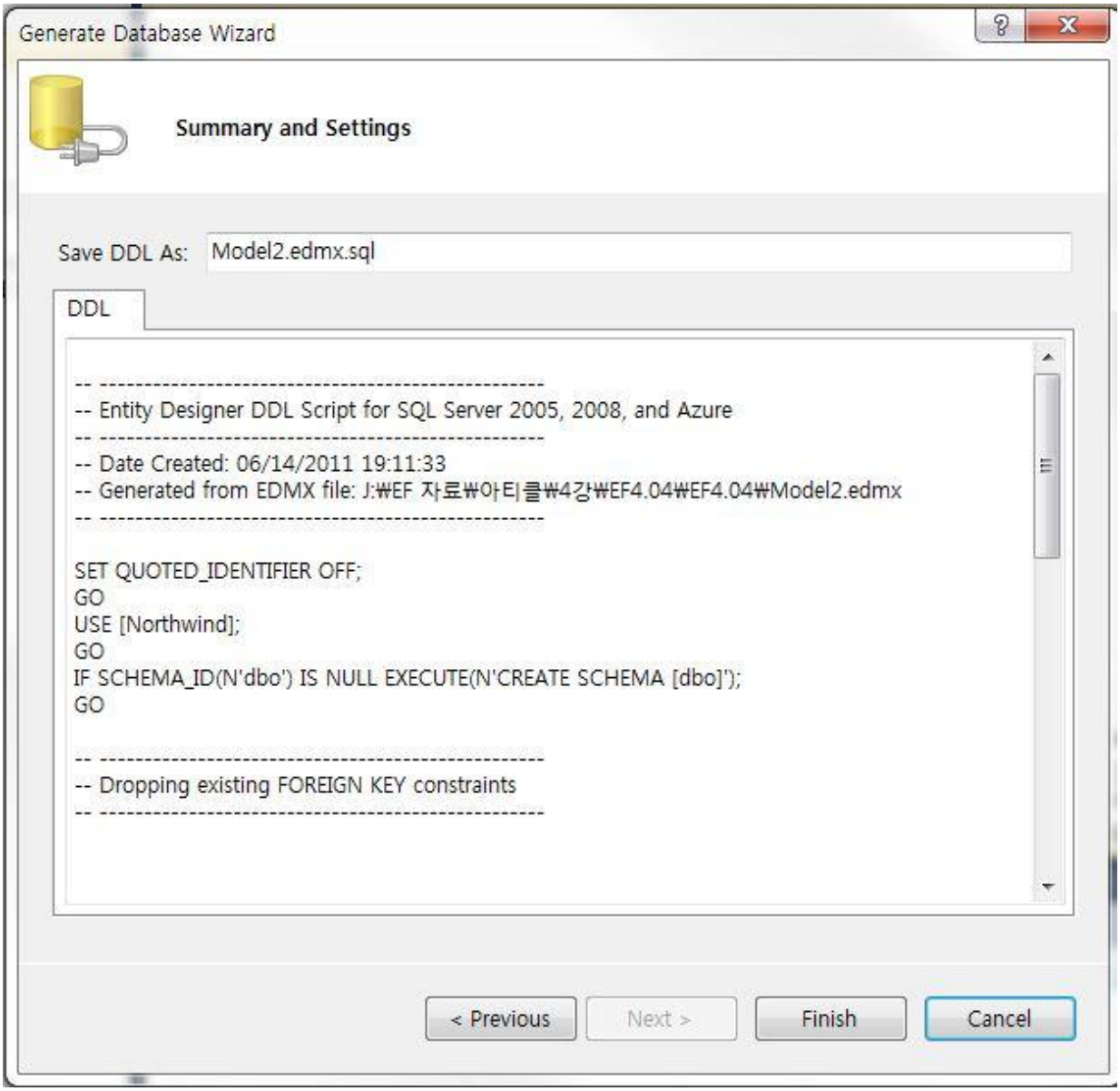
[그림 4-13] 모델 자동 생성

데이터 저장소를 설정하는 화면이 출력된다. 적당한 값을 선택하고"Next"을 클릭해준다.



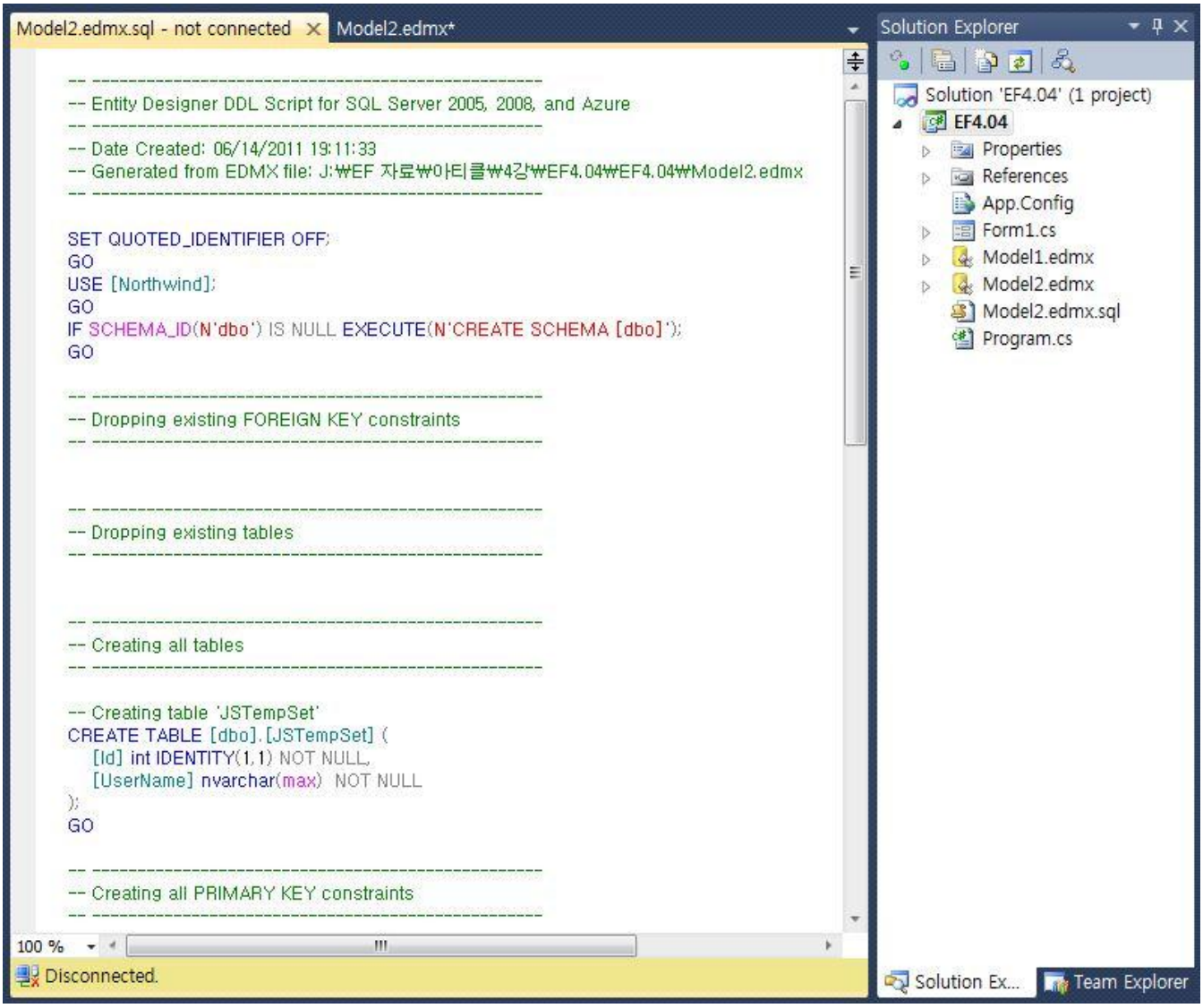
[그림 4-14] 데이터베이스 연결 설정

데이터베이스 개체 구조를 정의한 DDL(Data Definition Language)이 생성된다.



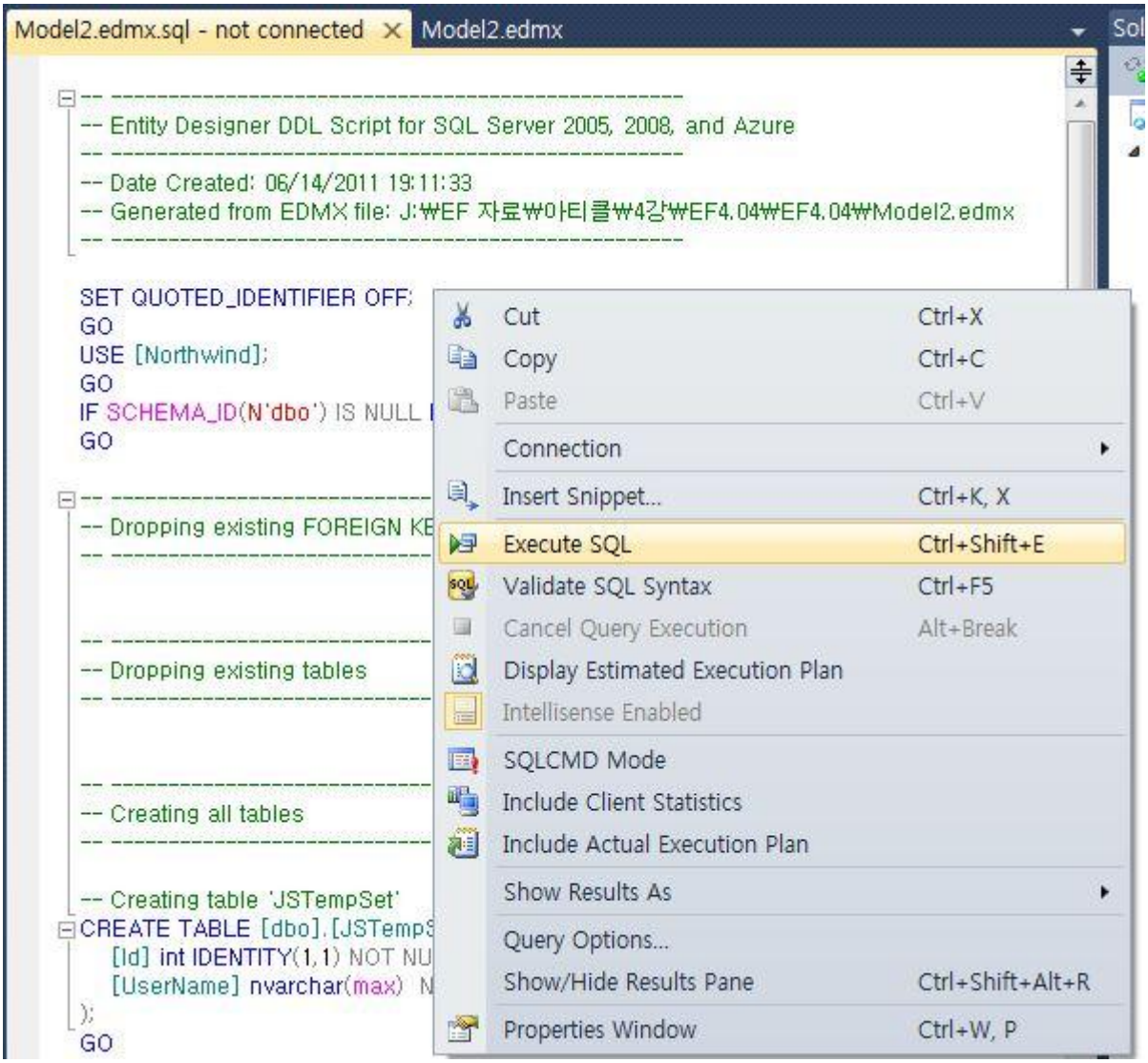
[그림 4-15] DDL 생성

완료를 클릭하면 스크립트가 화면에 출력된다.



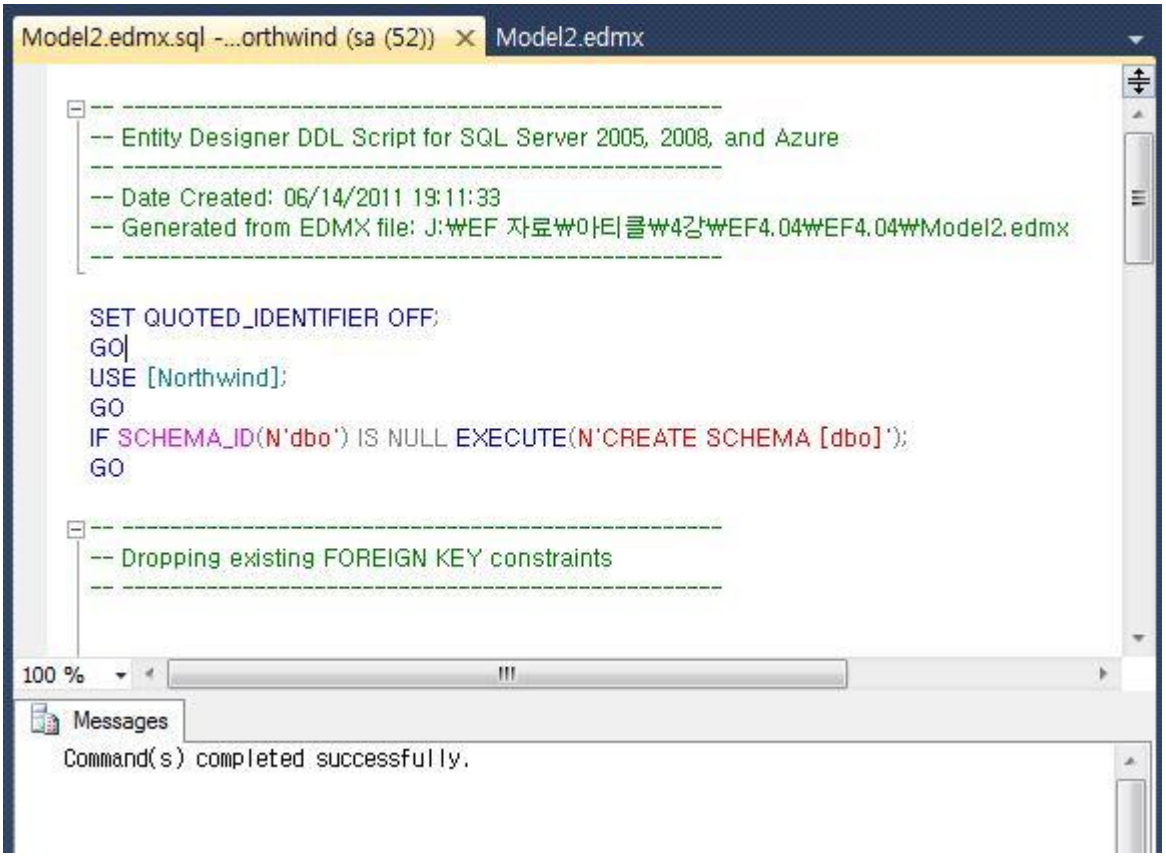
[그림 4-16] 스크립트

SQL 스크립트를 실행해보겠다.



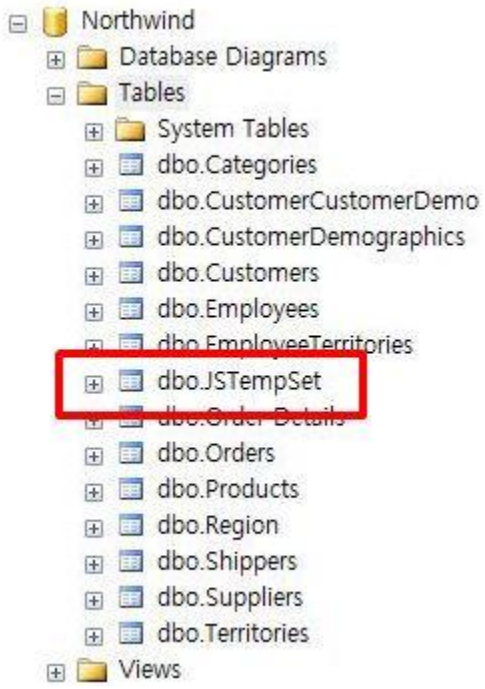
[그림 4-17] 스크립트 실행

아래 메세지 보니 성공적으로 완료 되었다. 실제로 생성되었는지 데이터베이스를 확인해보겠다.



[그림 4-18] 스크립트 실행 결과

오호~ 생성되었네요~



[그림 4-19] 데이터베이스

개념 모델 이후에 저장소에 개체를 생성하는 방법을 Model-First 이라고 한다.

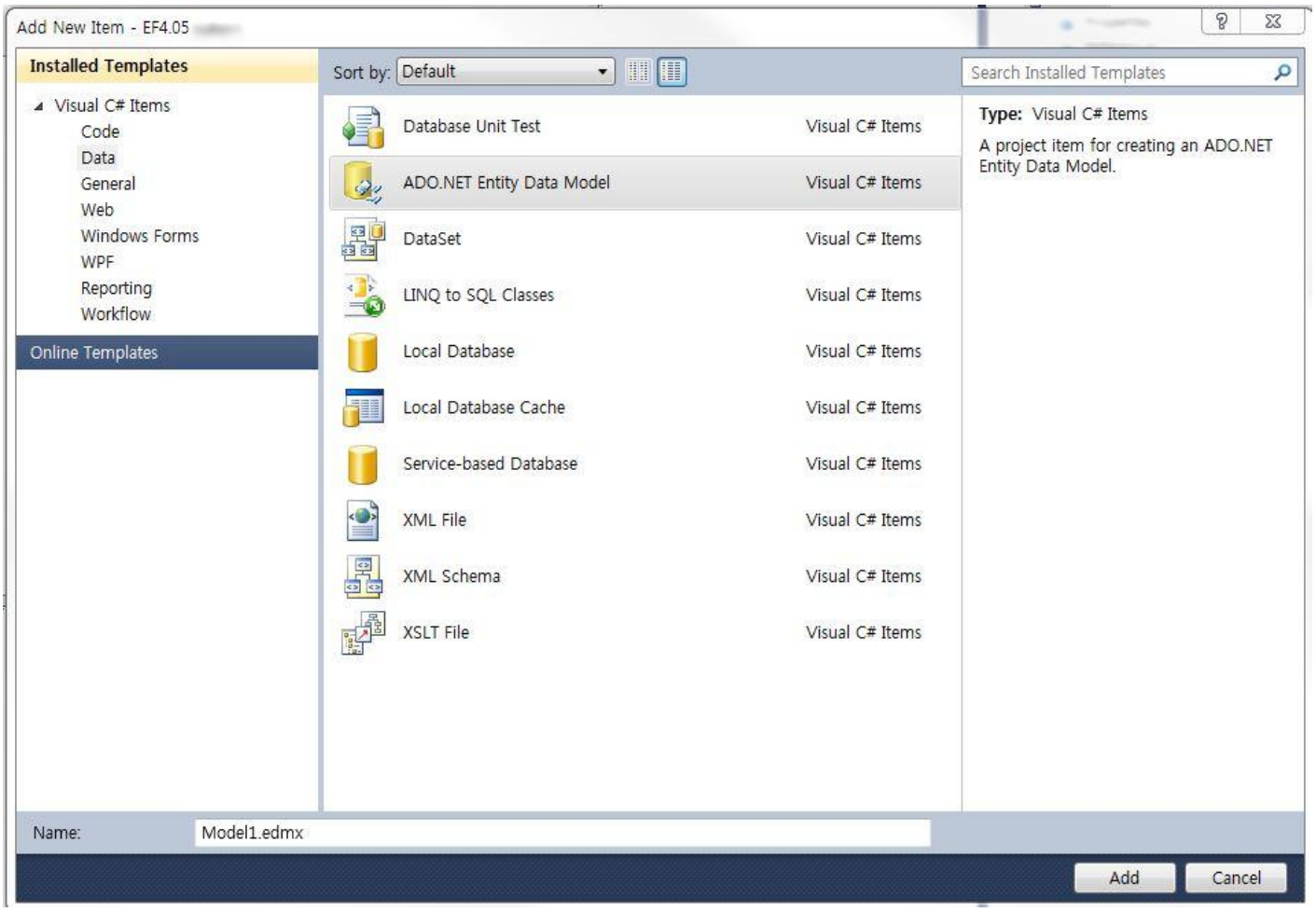
[Entity Framework 강좌] 05. Entity Framework Context

Entity Framework Context

이전 세션에서 Database 에서 개체들을 Entity 화하는 부분을 확인해보았다. 이번 시간은 이 엔티티를 가지고 실제 컨트롤에 데이터를 바인딩하는 부분을 살펴보겠다.

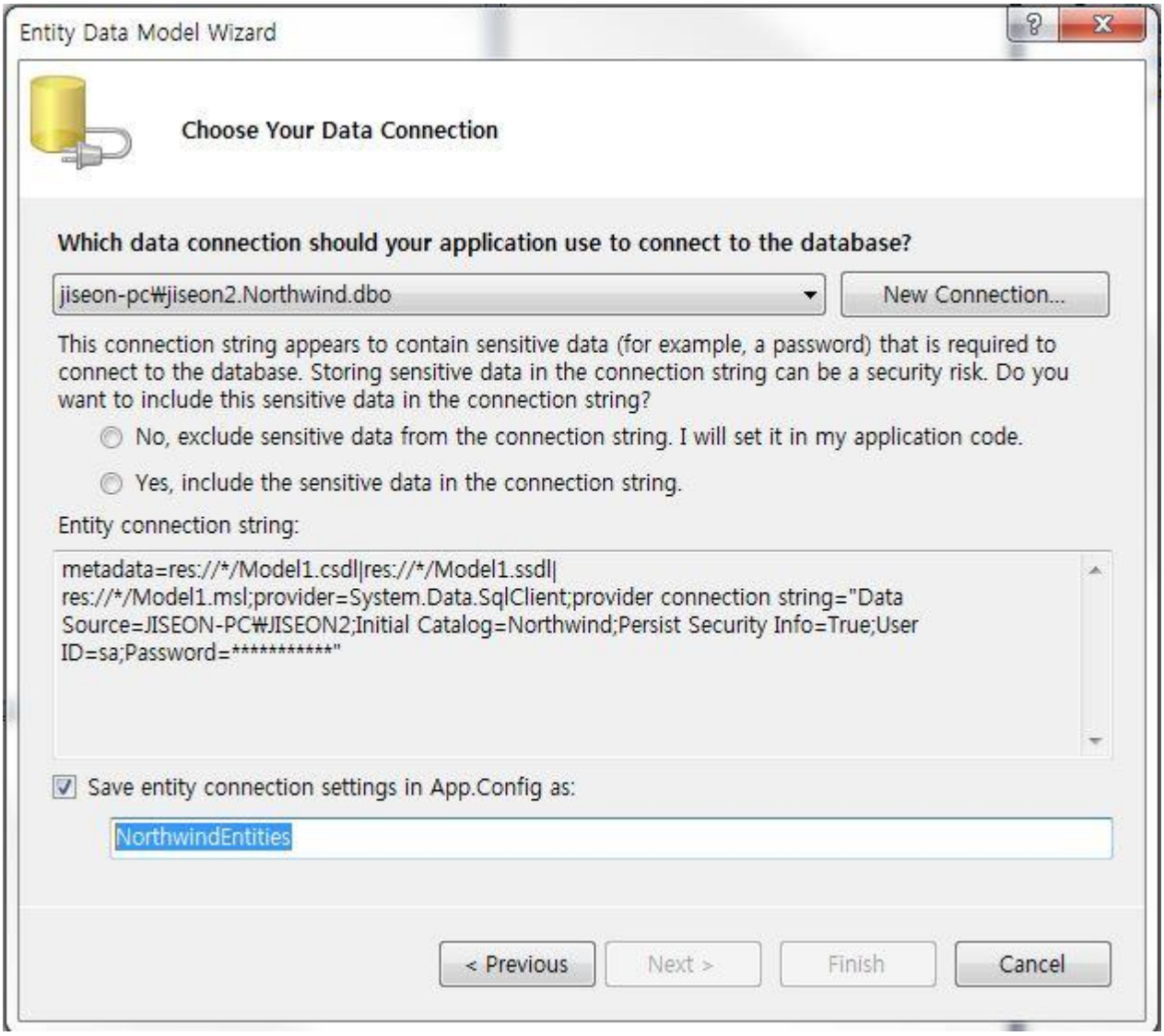
<실전! 따라해보기!!>

EDM 항목을 하나 추가해준다.



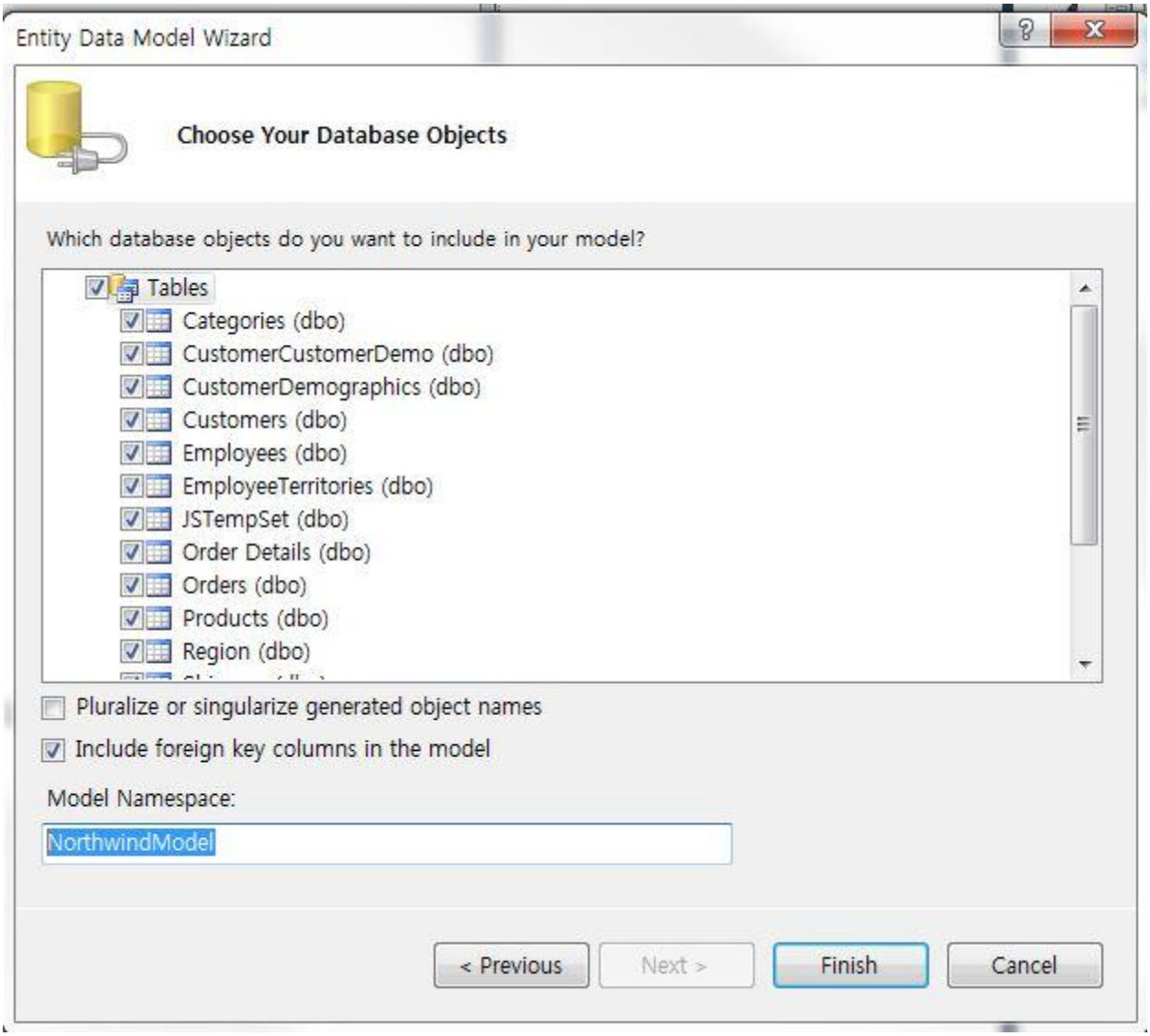
[그림 5-1] EDM 항목 추가

데이터베이스를 연결해준다. 저는 기본인 Northwind 를 연결하겠다.



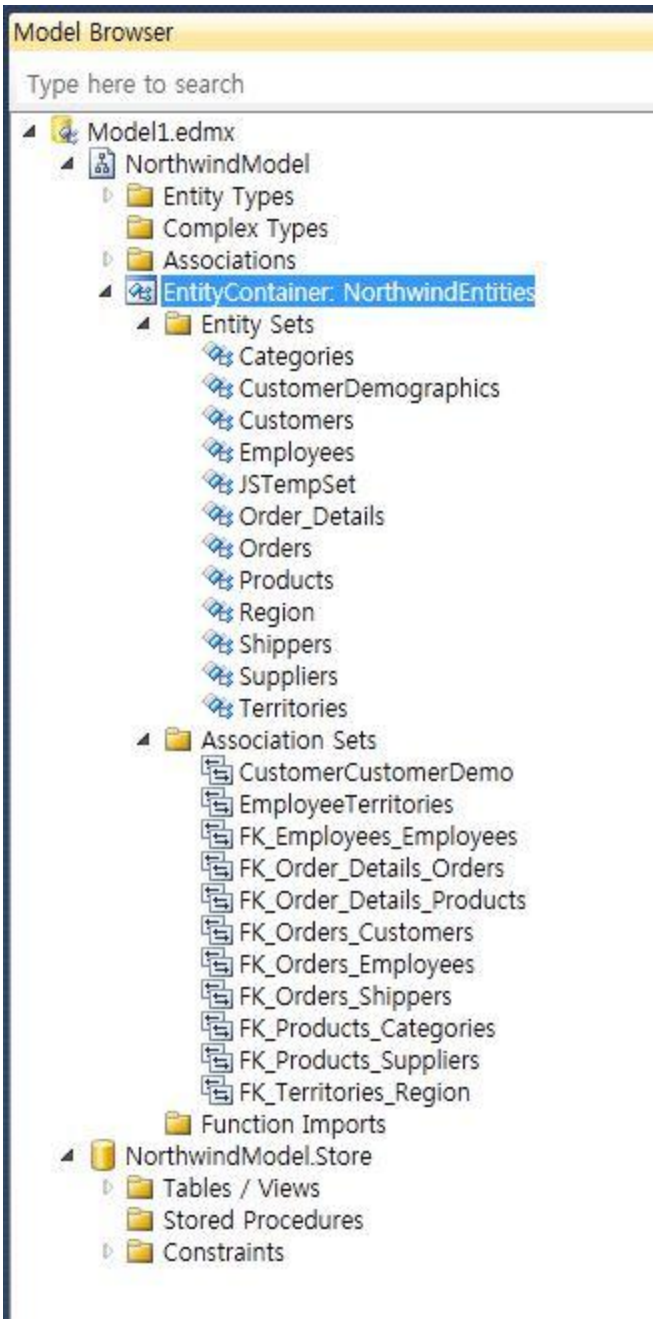
[그림 5-2] EDM Database 연결

모델 네이스페이스를 지정해 준다.



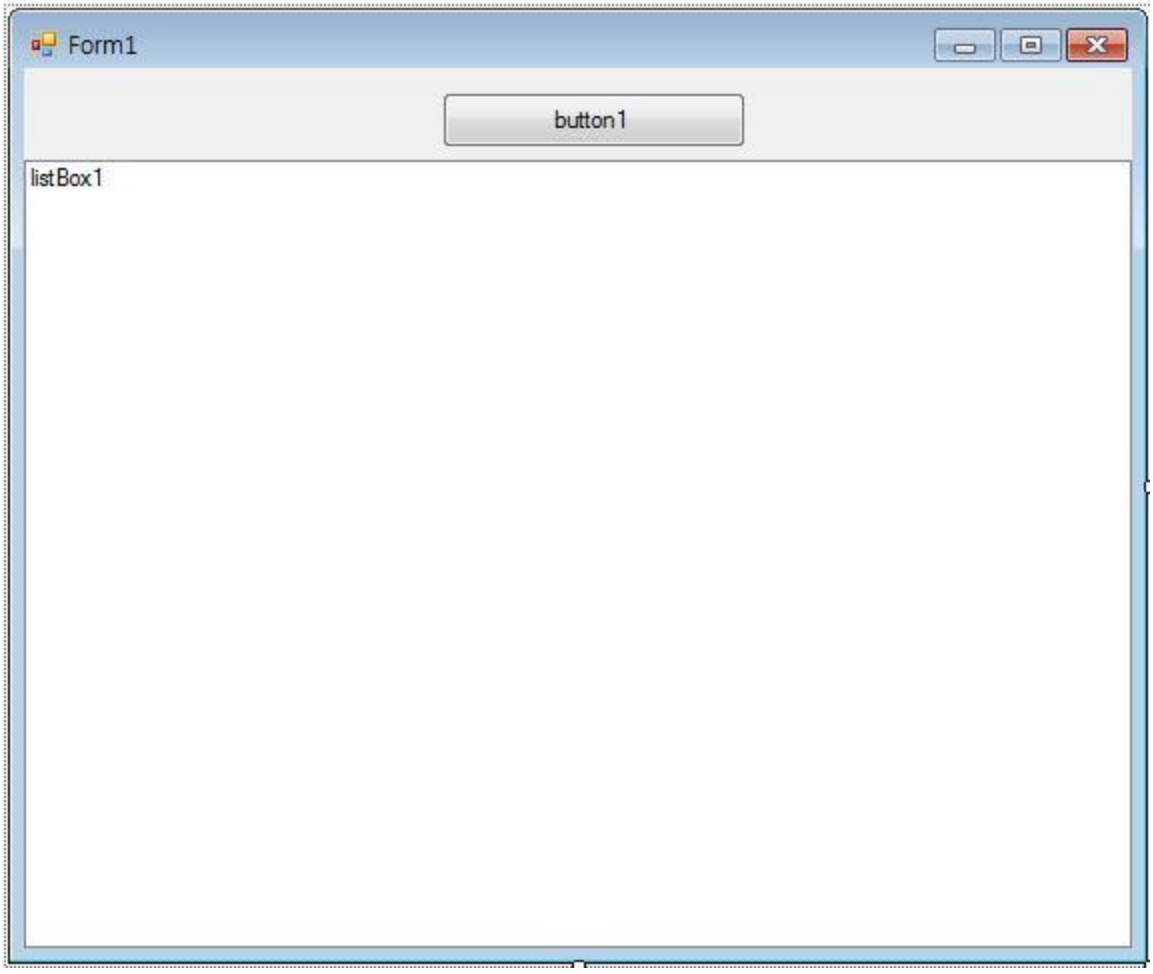
[그림 5-3] Database 개체 선택

EDM 이 자동생성 된 것을 확인하셨죠?! 모델 브라우저에서 다음과 같이 NorthwindEntities 가 생성된 것이 보인다. EntityContainer 는 EntityFramework 의 ObjectContext 클래스를 상속한다. ObjectContext 은 정의된 EDM 엔티티 형식의 객체들의 데이터를 관리하는 기본 클래스이다.



[그림 5-4] 모델 브라우저

데이터를 뿌려줄 UI 를 간단히 구성해보겠다. 버튼과 리트스박스로 구성되었으며 버튼을 클릭하면 리스트박스에 데이터를 바인딩 하려한다.

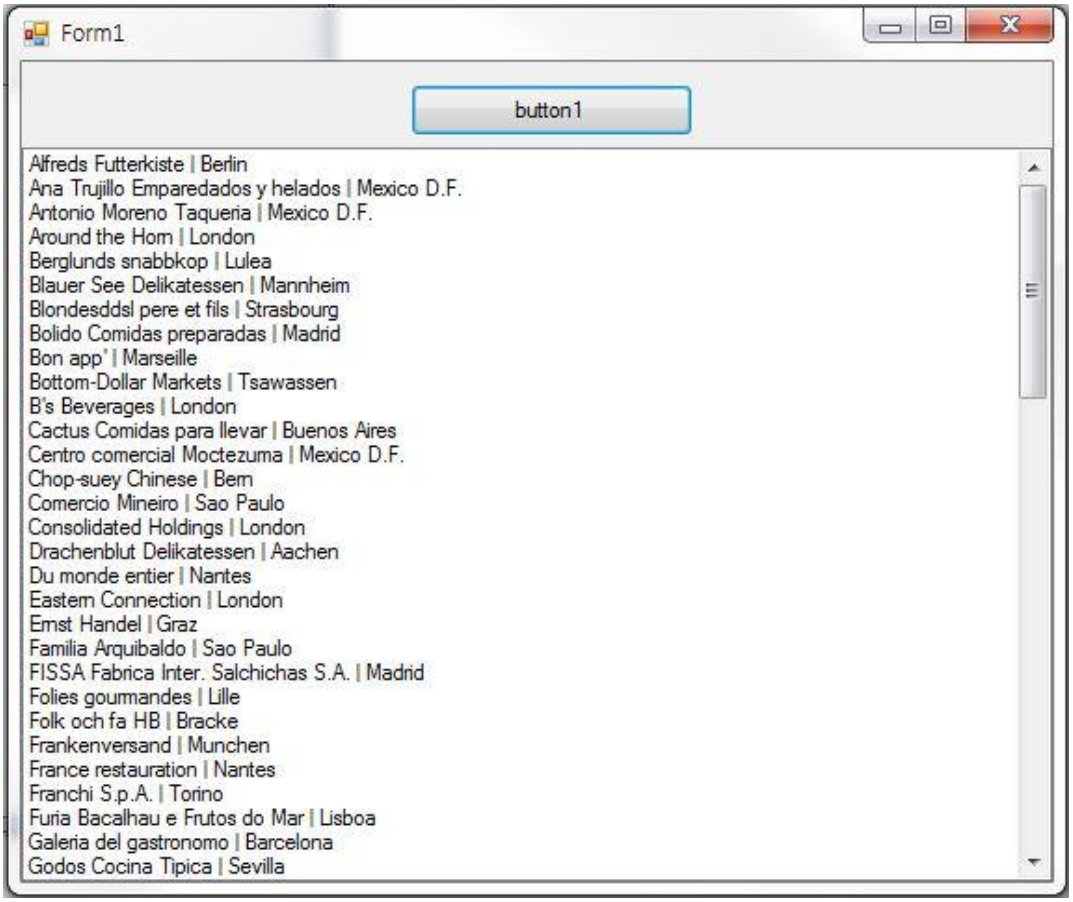


[그림 5-5] UI 구성

버튼 클릭 시 소스 코드는 다음과 같다. 앞에서 언급한 LINQ 로 Object 를 질의하시면 된다. NorthwindEntities 의 Customers 엔티티를 가져와서 실제 데이터베이스 데이터를 바인딩하겠다. 조건에 맞는 데이터를 가져와야 한다면 var customer = context.Customers;➔ 에서 조건 질의 하면 된다.

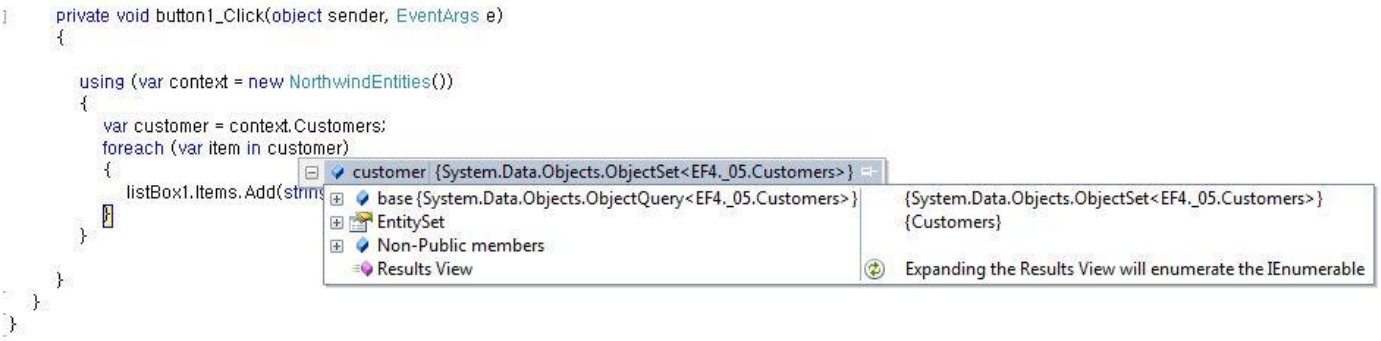
```
private void button1_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        var customer = context.Customers;
        foreach (var item in customer)
        {
            listBox1.Items.Add(string.Format("{0} | {1}", item.CompanyName, item.City));
        }
    }
}
```

프로그램을 실행해서 버튼을 클릭해보겠다. 다음과 같이 목록이 주루룩~ 조회되는 것을 보실 수 있다.



[그림 5-6] UI Data Bind

디버깅으로 Customers 를 확인해보면 ObjectSet 객체가 반환되는 것을 확인 할 수 있다. 이렇게 EDM 에서 엔티티들을 ObjectSet 으로 구성하는 것을 확인 하였다.



[그림 5-7] ObjectSet

EDM 코드를 한번 더 살펴보죠. ObjectSet 으로 다음과 같이 반환을 하고 있다.

```
/// <summary>
/// No Metadata Documentation available.
/// </summary>
public ObjectSet<Customers> Customers
{
    get
    {
        if ((_Customers == null))
        {
            _Customers = base.CreateObjectSet<Customers>("Customers");
        }
        return _Customers;
    }
}
private ObjectSet<Customers> _Customers;
```

<추가 팁!>

SQL Server 에서는 논리적으로 다음과 같은 순서로 질의한다.

(8) SELECT (9) TOP

(1) FROM

(3) JOIN

(2) ON

(4) WHERE

(5) GROUP BY

(6) WITH

(7) HAVING

(1) ORDER BY

소스파일 : EF4.05.zip

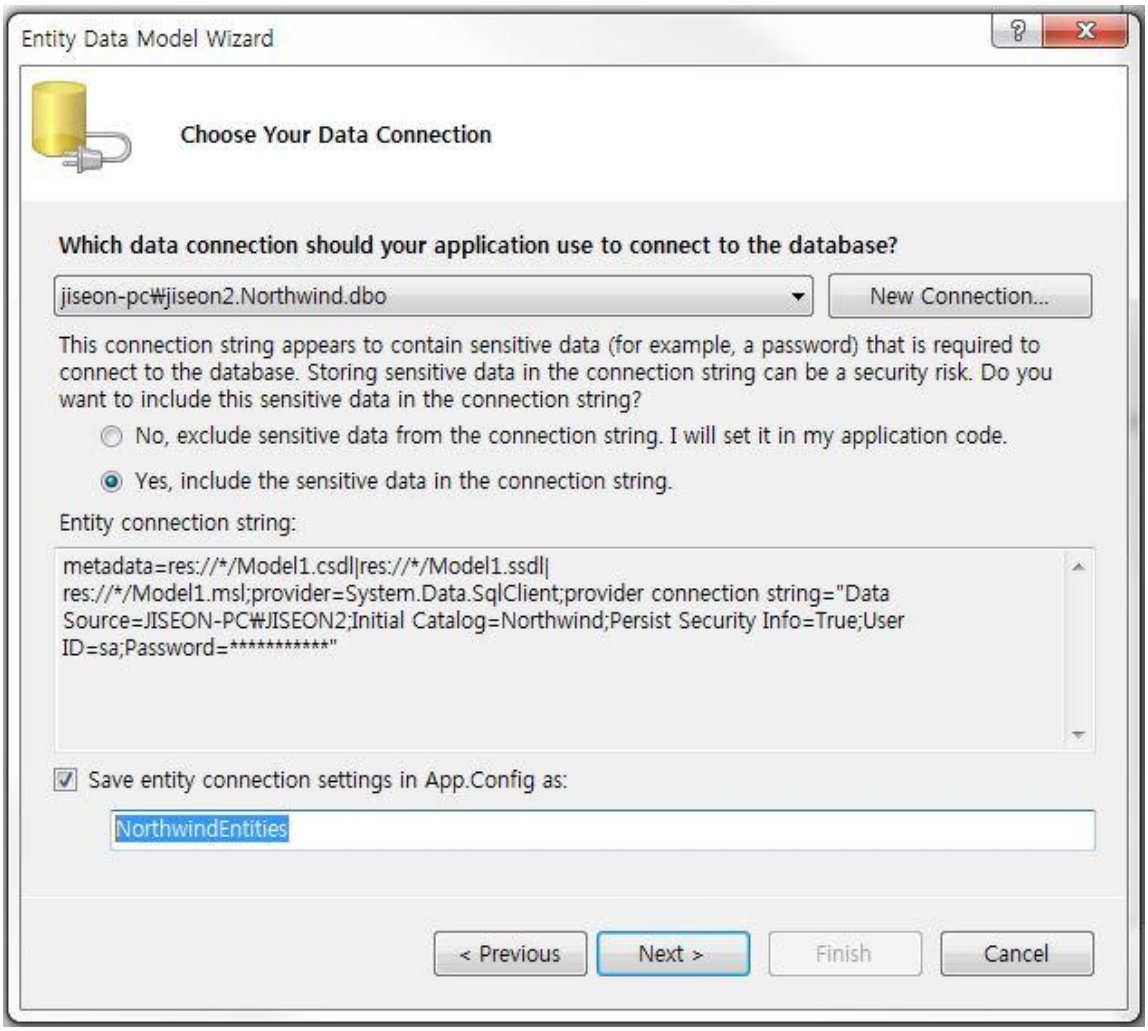
[Entity Framework 강좌] 06. Entity Framework – Entity CRUD

Entity Framework - Entity CRUD

이번에 알아 볼 내용은 Entity Framework 의 실지적인 Entity CRUD 이다. 하는 일은 데이터를 가져와서 데이터를 생성하고 조회하고 수정하고 삭제하는 등 개발에 필수적으로 필요한 작업이다. 이번에 엔티티를 가지고 와서 작업을 해보는 것을 살펴보겠다.

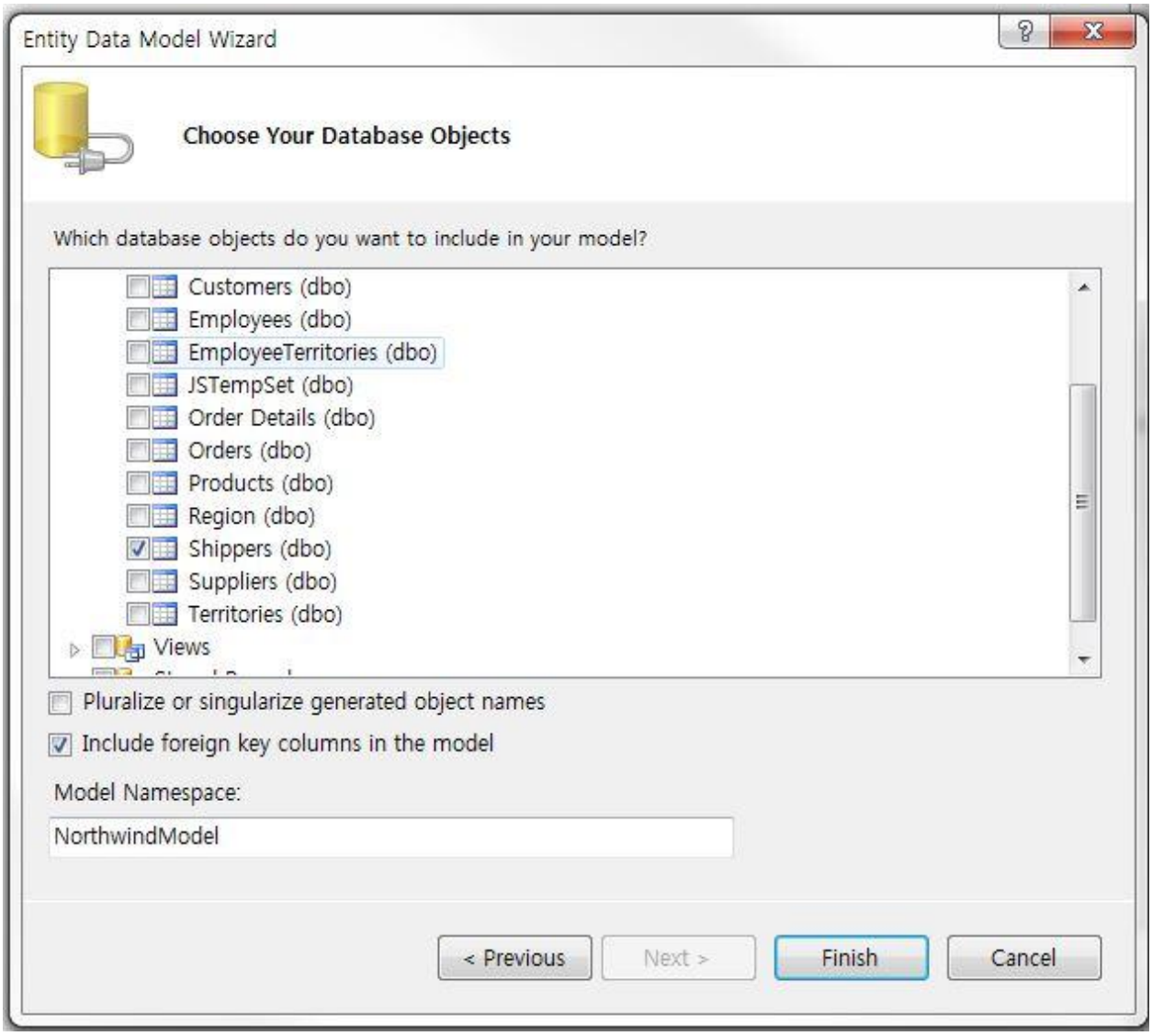
<Entity 조회(Select)>

엔티티 조회하는 것은 이미 앞 세션에서 설명을 들였다. 반복학습이 될 거라 생각된다. 우선 Northwind 데이터베이스를 연결하겠다.



[그림 6-1] DB 연결

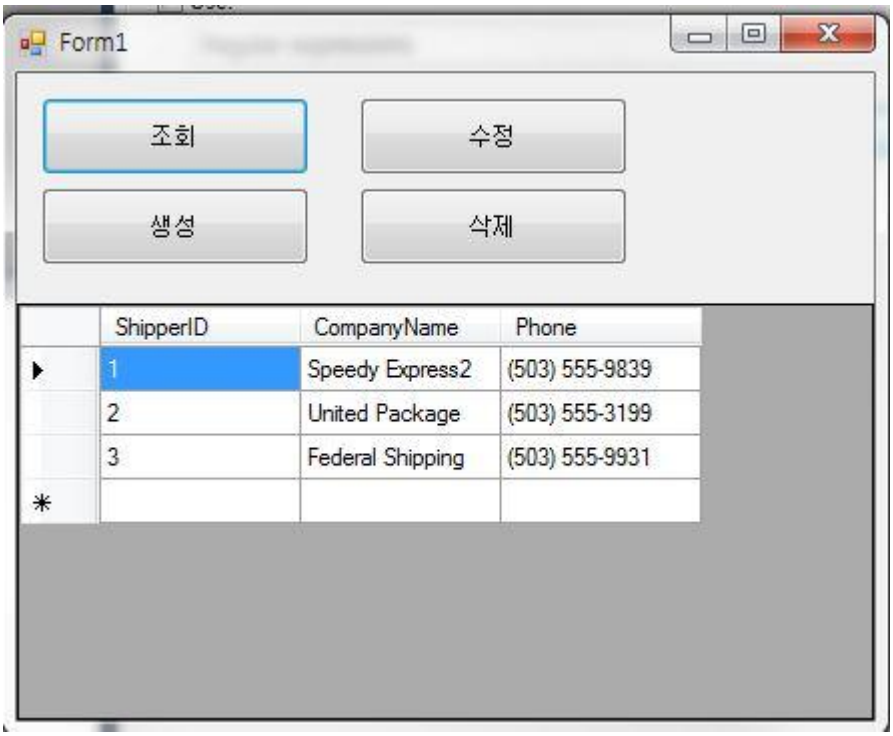
저희가 간단히 CRUD 기능을 살펴볼 예정이니 테이블 하나만 선택하겠다. Shippers 테이블을 선택해준다.



[그림 6-2] 테이블 선택

자 엔티티를 가져왔다면 데이터를 조회하는 코드를 작성해보겠다. 개체 컨텍스트를 가져와서 그리드뷰에 뿌려준다. 컨텍스트에서 Shippers 엔티티를 조회하면 된다.

```
private void btnSelect_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        dataGridView1.DataSource = context.Shippers;
    }
}
```

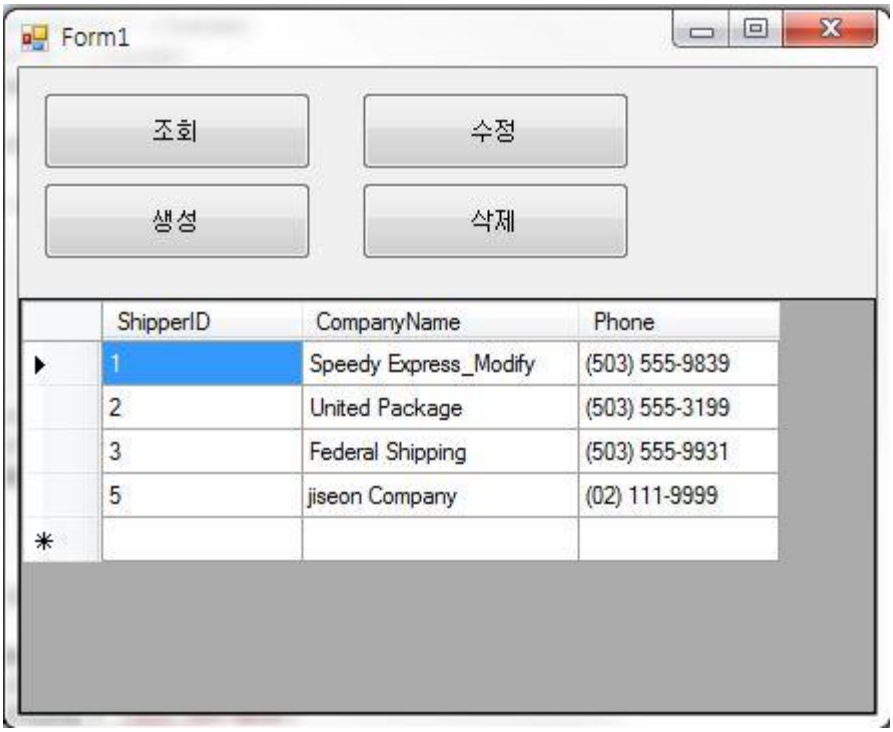


[그림 6-3] 조회화면

<Entity 수정(Update)>

다음은 엔티티 수정이다. 수정하려는 데이터를 조회해서 해당 엔티티 속성들 값을 수정한다. 수정 후에 데이터베이스에 반영이 되려면 최종적으로 SaveChanges 메서드를 호출해줘야 한다. SaveChanges 는 데이터 소스인 데이터베이스와 엔티티 컨텍스트 사이에서 모든 업데이트 내용을 유지하면 추적 기능을 제공하고 있다.

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        var shipper = context.Shippers.Where(p => p.ShipperID == 1).First();
        shipper.CompanyName = shipper.CompanyName + "_Modify";
        shipper.Phone = "(503) 555-9839";
        context.SaveChanges();

        dataGridView1.DataSource = context.Shippers;
    }
}
```



[그림 6-4] 수정버튼 후 화면

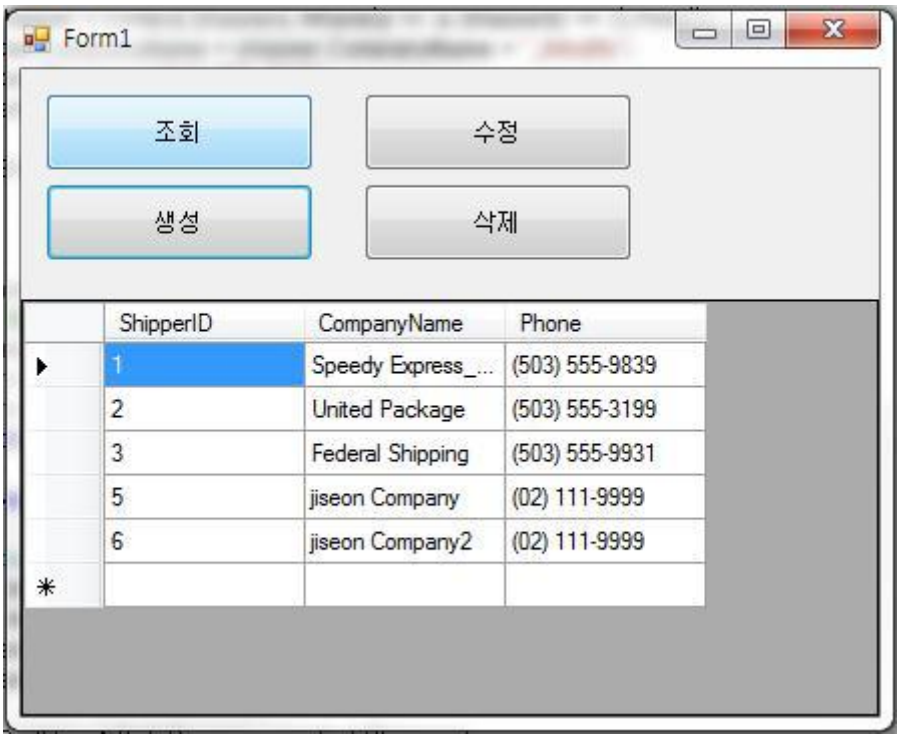
<Entity 추가(Insert)>

엔티티 추가이다. Shippers 엔티티 구조를 먼저 살펴보겠다. ShipperID 컬럼이 자동 증가이다. 데이터 추가시에는 `CompanyName, Phone` 컬럼의 데이터만 매핑해주면 된다.

```
CREATE TABLE [dbo].[Shippers](
        [ShipperID] [int] IDENTITY(1,1) NOT NULL,
        [CompanyName] [nvarchar](40) NOT NULL,
        [Phone] [nvarchar](24) NULL,
 CONSTRAINT [PK_Shippers] PRIMARY KEY CLUSTERED
(
        [ShipperID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Shippers 객체를 하나 생성해서 데이터를 매핑해줍니다. 컨텍스트 AddObject 메서드를 통해서 Shippers 엔티티 데이터를 생성한다. 데이터베이스 반영을 위해서는 SaveChanges 메서드를 꼭 출해줘야 한다.

```
private void btnCreate_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        Shippers addEntity = new Shippers();
        addEntity.CompanyName = "jiseon Company2";
        addEntity.Phone = "(02) 111-9999";
        context.Shippers.AddObject(addEntity);
        context.SaveChanges();

        dataGridView1.DataSource = context.Shippers;
    }
}
```

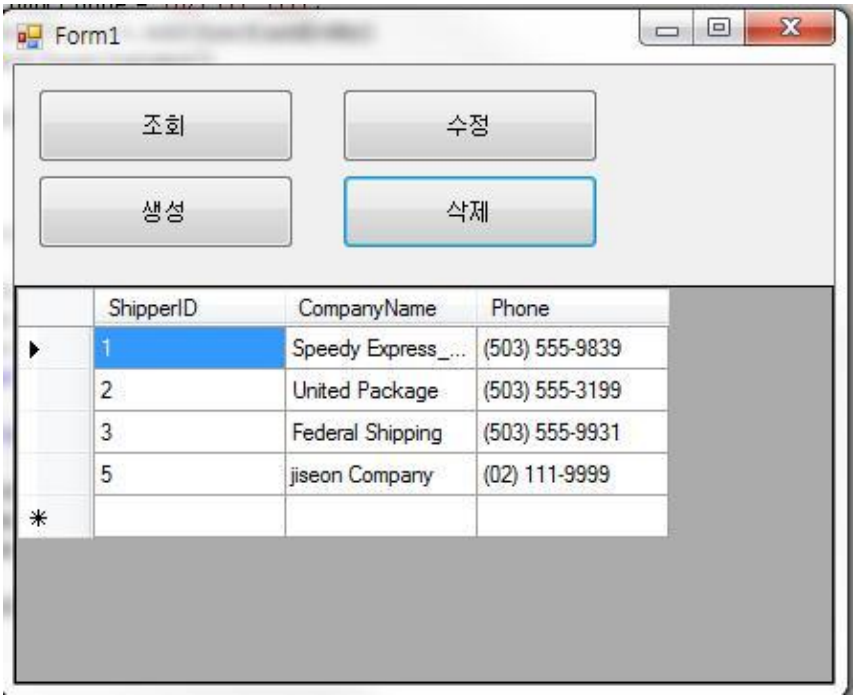


[그림 6-5] 엔티티 생성 후 화면

<Entity 삭제(Delete)>

마지막으로 엔티티 삭제이다. 엔티티 수정과 마찬가지로 특정 데이터를 조회해서 컨텍스트 DeleteObject 에 삭제 엔티티를 할당한다. SaveChanges 메서드를 호출하면 실제 데이터베이스에서 해당 데이터가 삭제된다.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        var shipper = context.Shippers.Where(p => p.ShipperID == 6).First();
        context.DeleteObject(shipper);
        context.SaveChanges();

        dataGridView1.DataSource = context.Shippers;
    }
}
```



[그림 6-6] 삭제 후 화면

엔티티를 조회하고 수정, 신규생성, 삭제하는 것을 살펴보았다.

소스 파일 : EF4.06.zip

[Entity Framework 강좌] 07. Entity Framework – Entity Stored Procedure 활용(1)

Entity Framework - Entity Stored Procedure 활용(1)

앞 세션에서 Entity Framework 의 Entity CRUD 작업에 대해서 알아보았다. 앞 세션에서 엔티티 작업은 실제로 Ad-Hoc Query 를 사용한다. 이번에는 Stored Procedure 를 사용하여 데이터 작업하는 것을 확인해보겠다.

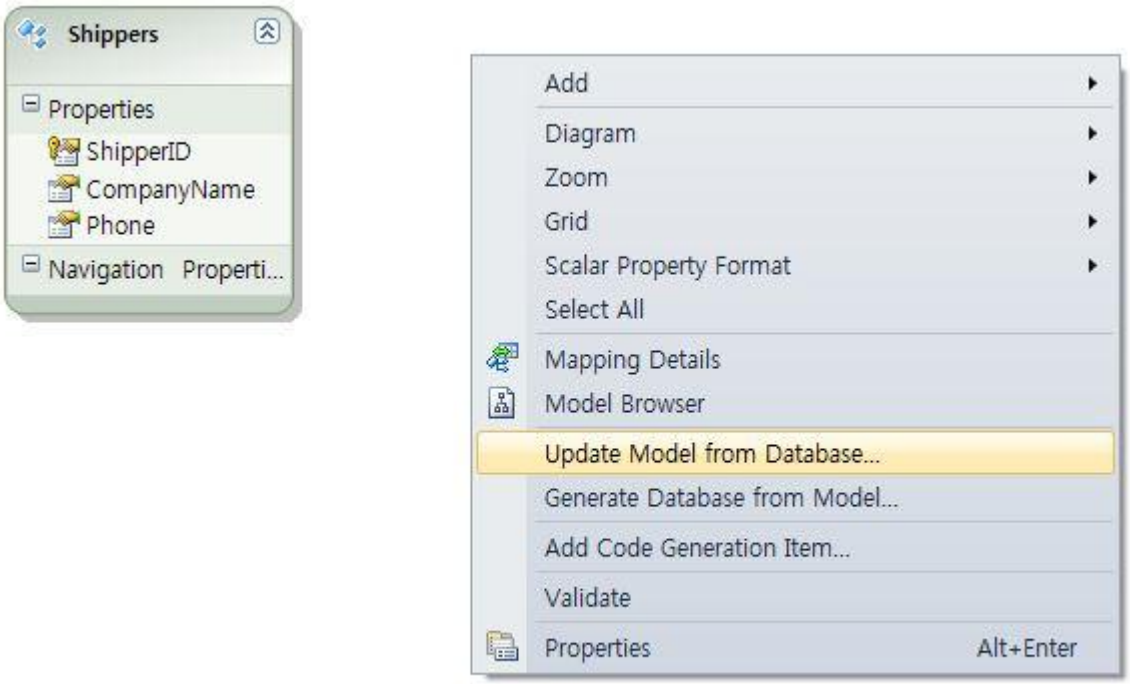
<기본 Stored Procedure 생성>

이번 내용을 확인해보기 위해 필요한 프로시져 스크립트이다. `Northwind` 데이터베이스에 생성해준다.

```sql
USE Northwind

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Description:     신규생성
-- =============================================
CREATE PROCEDURE Shippers_INSERT
       @CompanyName nvarchar(40),
       @Phone nvarchar(24)
AS
BEGIN
       SET NOCOUNT ON;

       INSERT INTO [Northwind].[dbo].[Shippers]
           ([CompanyName],[Phone])
     VALUES
           (@CompanyName, @Phone)

END
GO


-- =============================================
-- Description:     수정
-- =============================================
CREATE PROCEDURE Shippers_UPDATE
       @ShipperID INT,
       @CompanyName nvarchar(40),
       @Phone nvarchar(24)
AS
BEGIN
       SET NOCOUNT ON;

       UPDATE [Northwind].[dbo].[Shippers]
       SET [CompanyName] = @CompanyName, [Phone] = @Phone
       WHERE ShipperID = @ShipperID

END
GO


-- =============================================
-- Description:     삭제
-- =============================================
CREATE PROCEDURE Shippers_DELETE
       @ShipperID INT
AS
BEGIN
       SET NOCOUNT ON;

       DELETE FROM [Northwind].[dbo].[Shippers]
       WHERE ShipperID = @ShipperID

END
GO
```

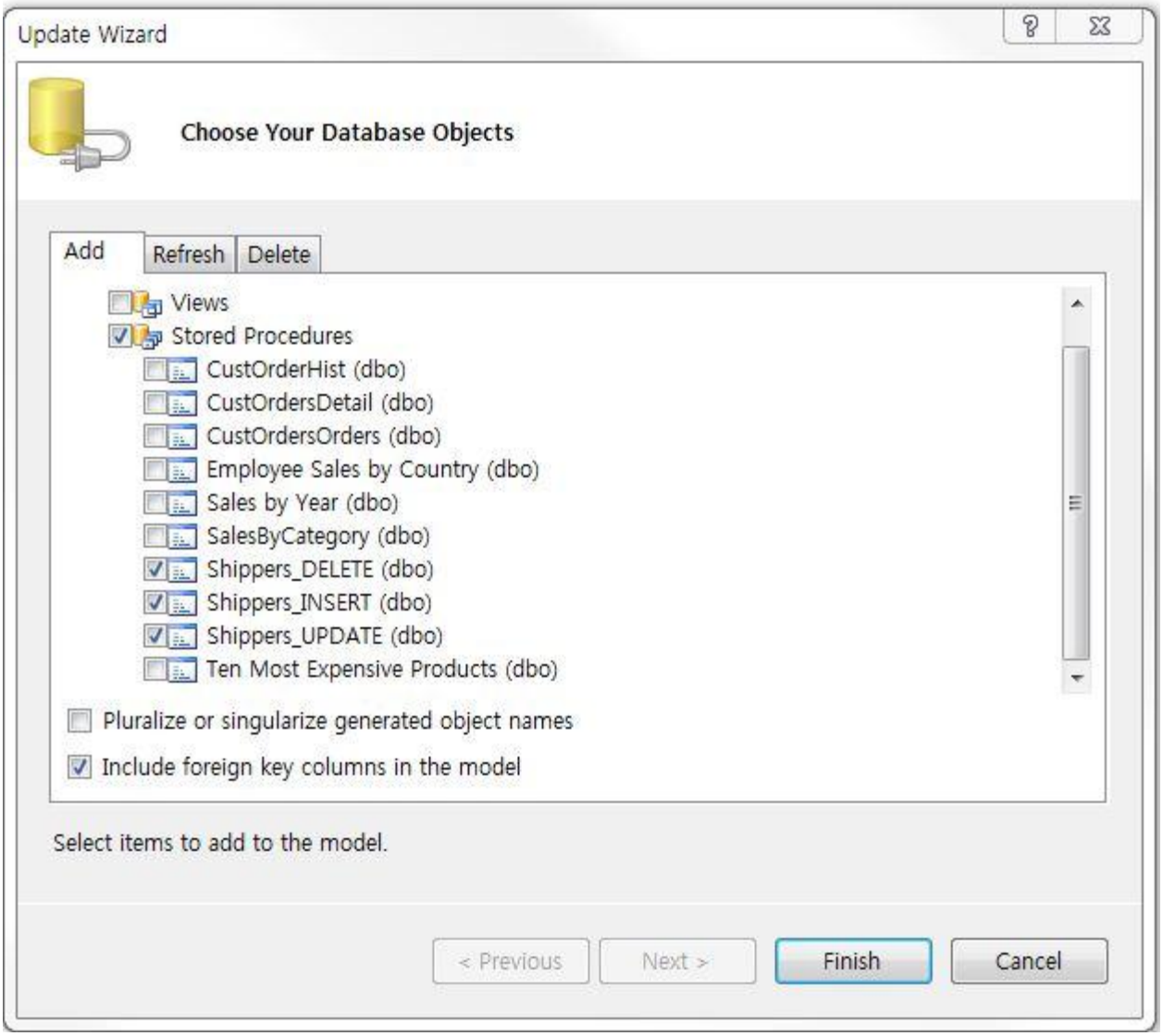
<Stored Procedure Entity 업데이트>

EDM 프로시져를 가져오겠다. "update Model from Database..."메뉴를 클릭해준다.



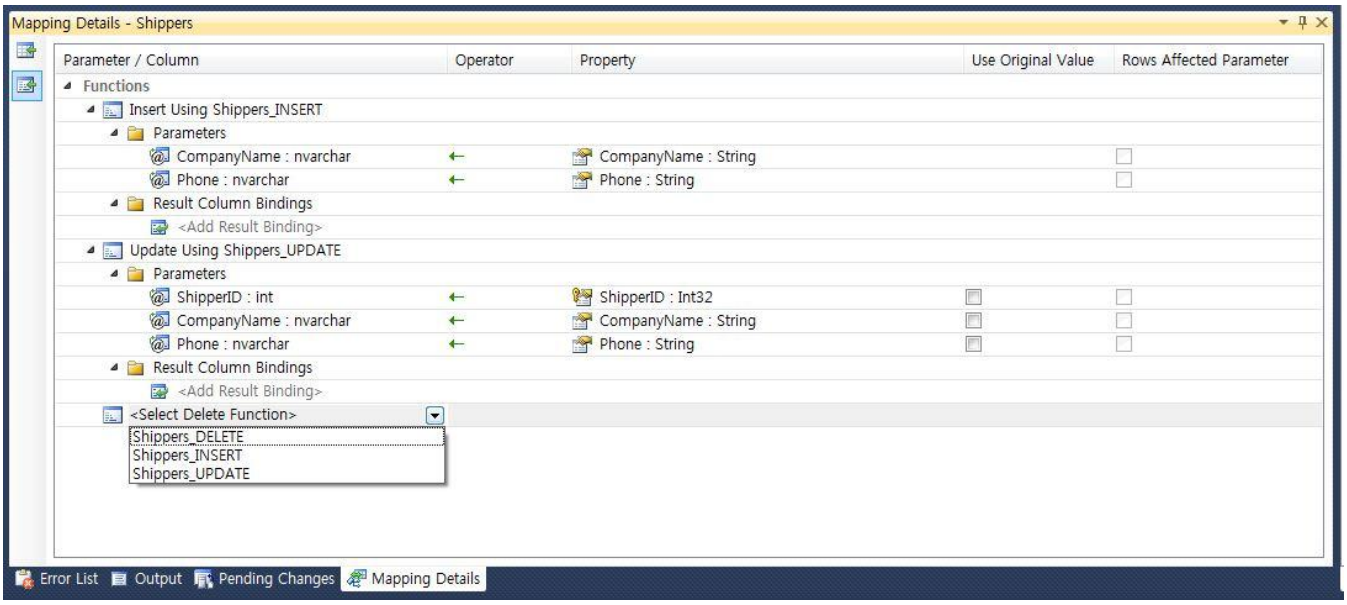
[그림 7-1] 모델 업데이트

앞에서 생성한 프로시져를 선택해준다.



[그림 7-2] Stored Procedure 추가

프로시져를 가져왔다면 엔티티 매핑 작업을 진행해준다. EDM 에서 매핑 상세에서 “Shippers” 엔티티의 프로시져 항목을 클릭하면 다음과 같은 화면이 나옵니다. Insert, Update, Delete 에 맞는 프로시져를 맞춰서 설정해주면 자동으로 매핑이 된다.



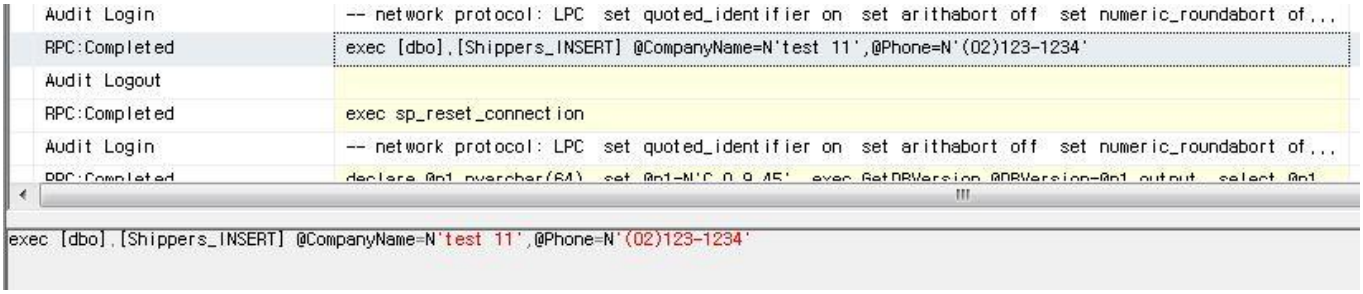
[그림 7-3] 엔티티 매핑

<Entity 생성(Insert)>

소스 코드를 보면 아시겠지만 앞 세션에서 다룬 엔티티 생성 코드와 동일하다.

```
private void btnInsert_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        Shippers shipper = new Shippers();
        shipper.CompanyName = "test 11";
        shipper.Phone = "(02)123-1234";
        context.Shippers.AddObject(shipper);
        context.SaveChanges();
        MessageBox.Show("record inserted");
    }
}
```

SQL Profiler 에서 확인하면 "Shippers_INSERT" 프로시져가 호출되어 데이터가 추가된 것을 확인 할 수 있다.



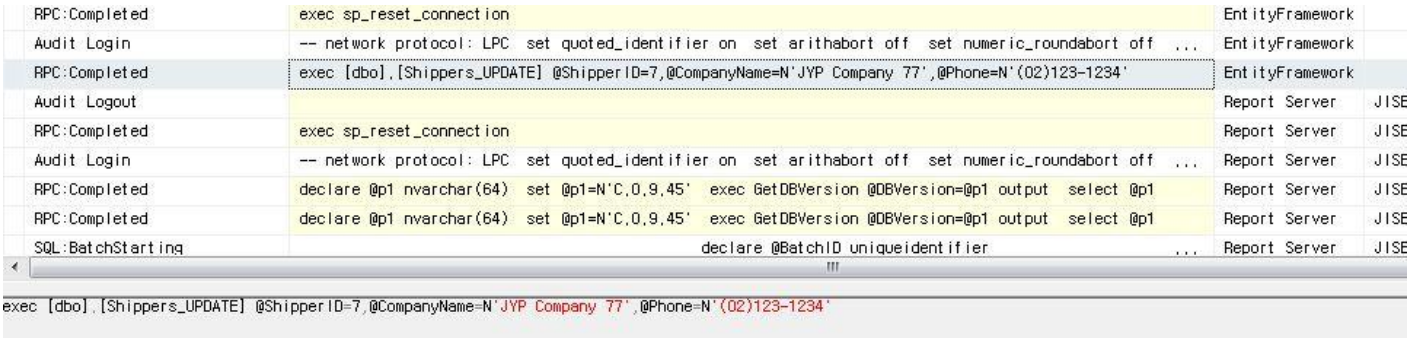
[그림 7-4] SQL Profiler 데이터 추가

<Entity 수정(Update)>

엔티티 데이터 수정도 마찬가지이다. 특정 항목에 대해서 데이터를 수정하고 SaveChange 메서드를 호출하여 최종적으로 데이터베이스에 반영하는 액션을 취한다.

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        var per = context.Shippers.Where(p => p.ShipperID == 7).First();
        per.CompanyName = "JYP Company 77";
        per.Phone = "(02)123-1234";
        context.SaveChanges();
        MessageBox.Show("record updated");
    }
}
```

SQL Profiler 에서 확인해보죠. "Shippers_UPDATE" 프로시져가 호출되고 있네요.



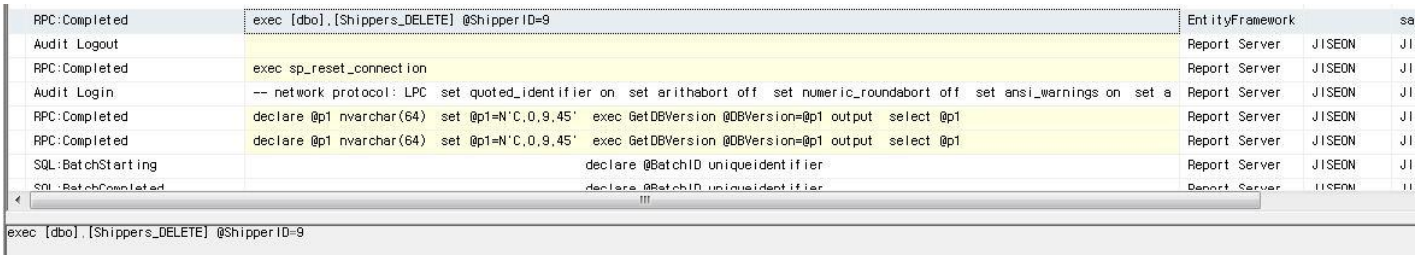
[그림 7-5] SQL Profiler 데이터 수정

<Entity 삭제(Delete)>

데이터 삭제하는 부분도 확인해보겠다.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        var shipper = context.Shippers.Where(p => p.ShipperID == 9).First();
        context.DeleteObject(shipper);
        context.SaveChanges();

        MessageBox.Show("record deleted");
    }
}
```

역시 "Shippers_DELETE" 프로시져가 호출되고 있다.



[그림 7-6] SQL Profiler 데이터 삭제

이번에 알아본 것은 간단히 요약하자만 특정 엔티티 기준에서 데이터 작업을 할 경우 Stored Procedure 를 사용하는 것을 확인해 봤다.

소스 코드 : EF4.07.zip

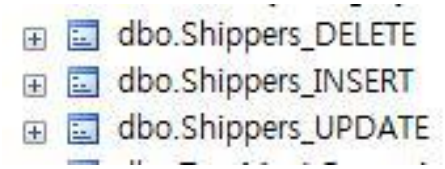
[Entity Framework 강좌] 08. Entity Framework – Entity Stored Procedure 활용(2)

Entity Framework - Entity Stored Procedure 활용(2)

Entity Framework 에서 Stored Procedure 를 다루는 방법에 대해서 앞 세션에서 다뤘는데 또 다른 방법이 있어 추가로 소개하고자 한다. Visual Studio 의 막강한 기능으로 대부분이 자동생성된다. 이번 시간도 아주 간단한 작업만으로 Entity Stored Procedure 다룰 수 있다.

<Stored Procedure>

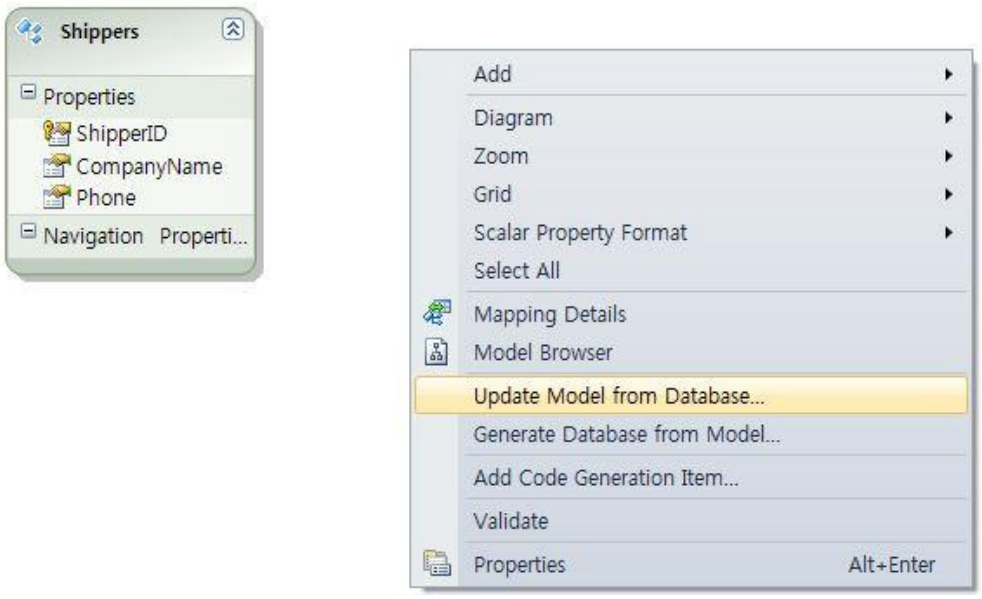
앞 세션에서 사용한 프로시져를 그대로 사용하겠다.



[그림 8-1] 프로시져 목록

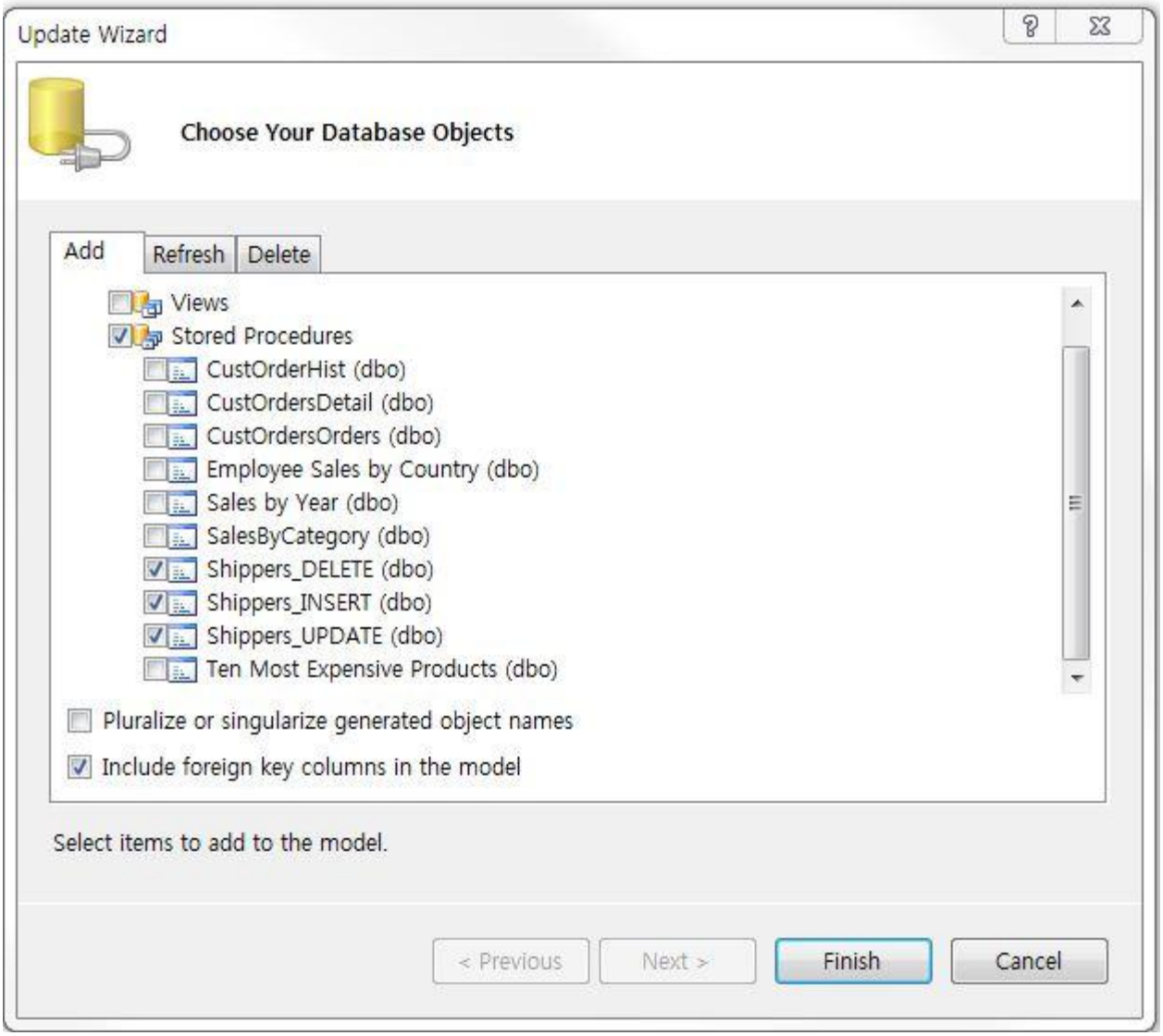
<Stored Procedure Entity 업데이트>

EDM 프로시져를 가져오겠다. "update Model from Database..."메뉴를 클릭해준다.



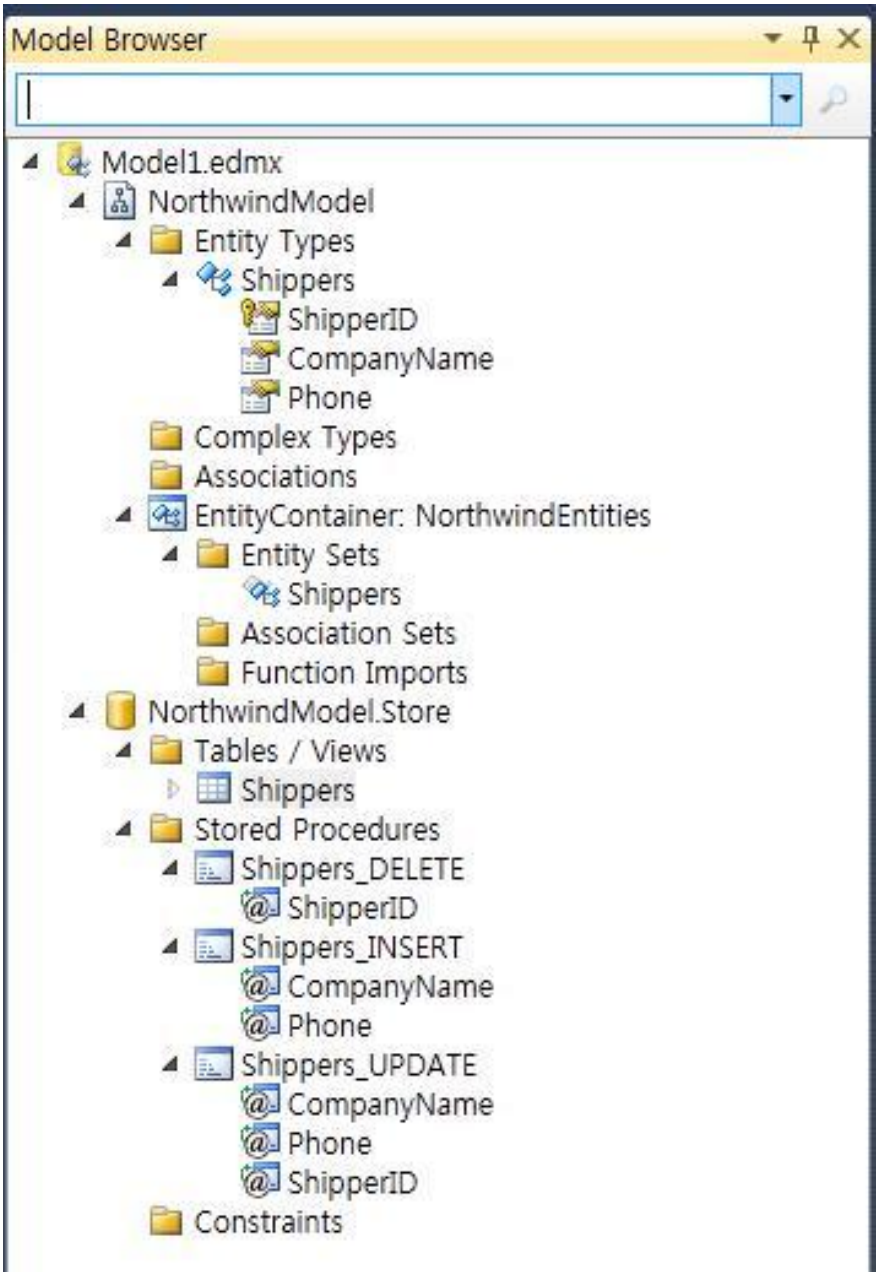
[그림 8-2] 모델 업데이트

추가할 해당 프로시져를 선택해준다.



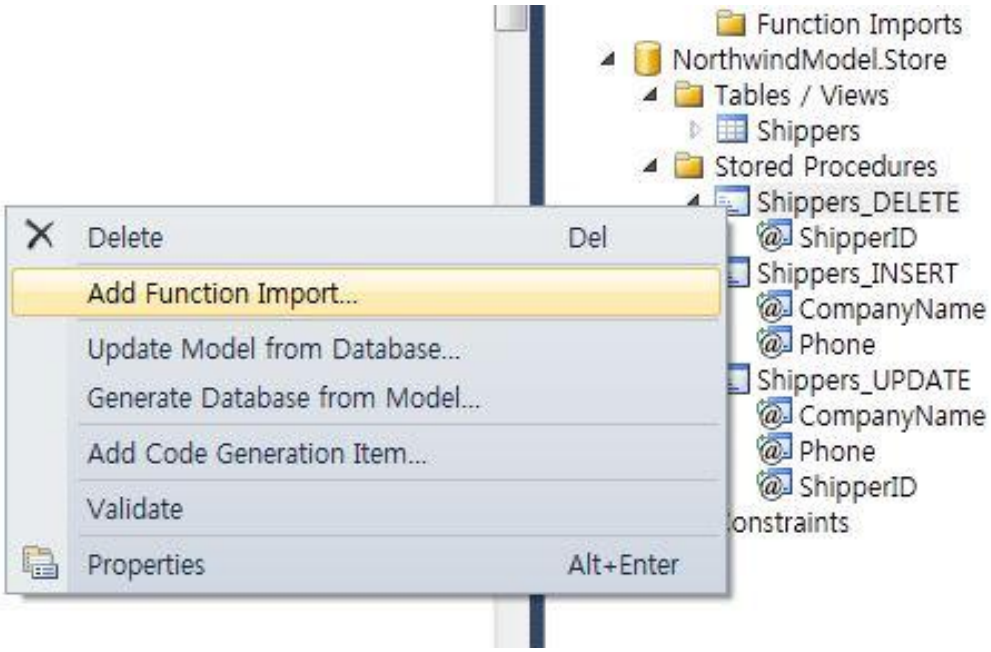
[그림 8-3] Stored Procedure 추가

기본적으로 엔티티 매핑 개체 추가 후 모델 브라우저를 보면 다음과 같이 트리가 구성된다.



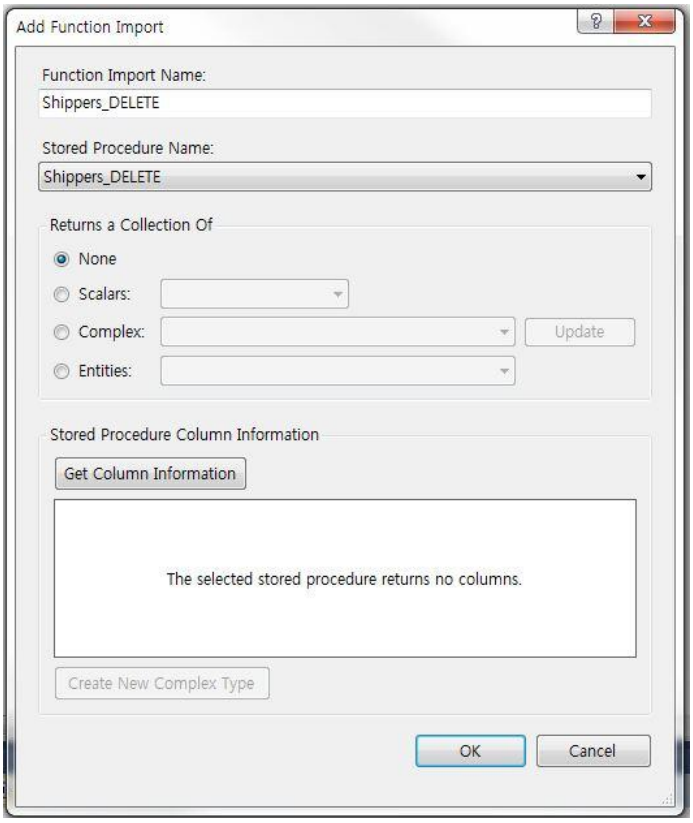
[그림 8-4] 모델 브라우저 트리

가지고 온 프로시져를 엔티티의 함수로 가져오는 작업을 추가 해줘야 한다. 프로시져에서 다음과 같이 "Add Function Import..."메뉴를 선택한다.



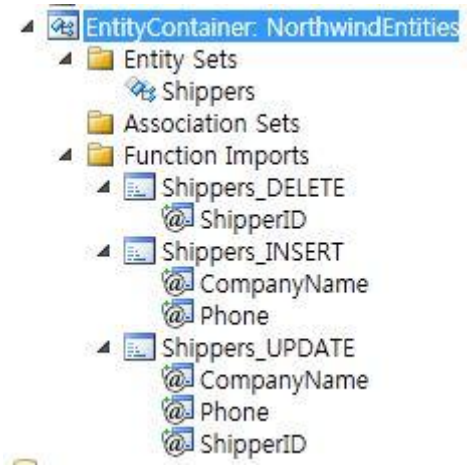
[그림 8-5] 모델 브라우저 트리 메뉴

함수 가져오는 것은 프로시져에 함수명만 지정해주면 해당 함수명으로 자동으로 매핑이 된다.



[그림 8-6] 함수 가져오기 설정

3 개 모두 함수 가져오기하면 엔티티의 함수 목록에 다음과 같이 출력된다. 이제 프로시져 가져오는 작업이 완료되었다.



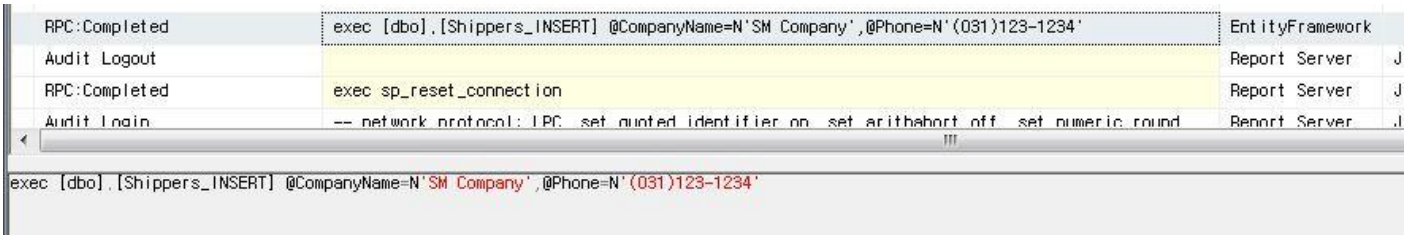
[그림 8-7] 모델 브라우저 트리 – 함수 가져오기

<Entity 생성(Insert)>

추가했던 함수명이 컨텍스트에서 메서드로 제공된다. 제공된 메서드에 맞춰 파라메터를 넣어주면 된다.

```
private void btnInsert_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        Shippers shipper = new Shippers();
        shipper.CompanyName = "test 11";
        shipper.Phone = "(02)123-1234";
        context.Shippers.AddObject(shipper);
        context.SaveChanges();
        MessageBox.Show("record inserted");
    }
}
```

SQL Profiler 에서 확인하면 "Shippers_INSERT" 프로시져가 호출되어 데이터가 추가된 것을 확인 할 수 있다.



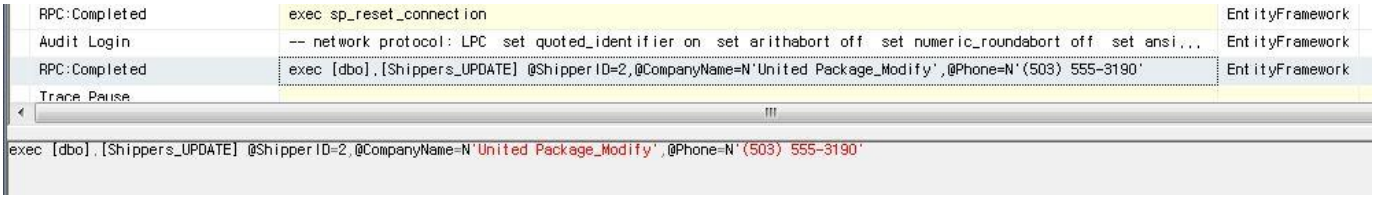
[그림 8-8] SQL Profiler 데이터 추가

<Entity 수정(Update)>

수정하는 "Shippers_UPDATE" 메서드를 호출하여 엔티티 수정을 한다.

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        context.Shippers_UPDATE(2, "United Package_Modify", "(503) 555-3190");
        MessageBox.Show("record updated");
    }
}
```

SQL Profiler 에서 확인하면 "Shippers_UPDATE" 프로시져가 호출되고 있는 것을 확인 할 수 있다.



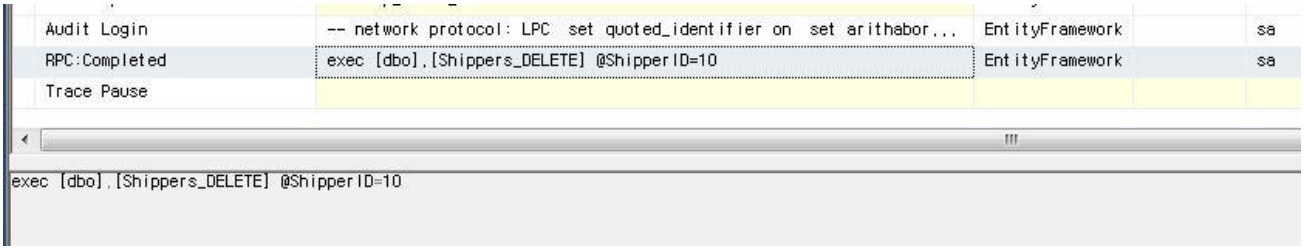
[그림 8-9] SQL Profiler 데이터 수정

<Entity 삭제(Delete)>

"Shippers_DELETE" 메서드를 호출하여 데이터 삭제를 한다.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    using (var context = new NorthwindEntities())
    {
        context.Shippers_DELETE(10);
        MessageBox.Show("record deleted");
    }
}
```

10 으로 파라메터를 할당한 "Shippers_DELETE" 프로시져가 호출되고 있다.



[그림 8-10] SQL Profiler 데이터 삭제

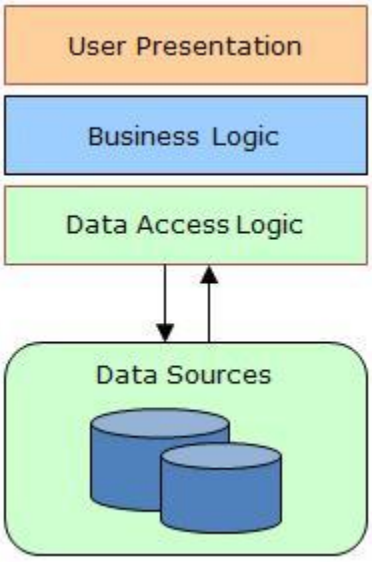
프로시져를 함수로 가져와서 메서드형태로 사용하는 것을 확인해보았다. 앞 세션에서 다룬 것은 테이블 기준으로 프로시져를 활용하는 것이었다.

소스 코드 : EF4.08.zip

[Entity Framework 강좌] 09. Entity Framework – ASP.NET MVC(1)

Entity Framework – ASP.NET MVC(1)

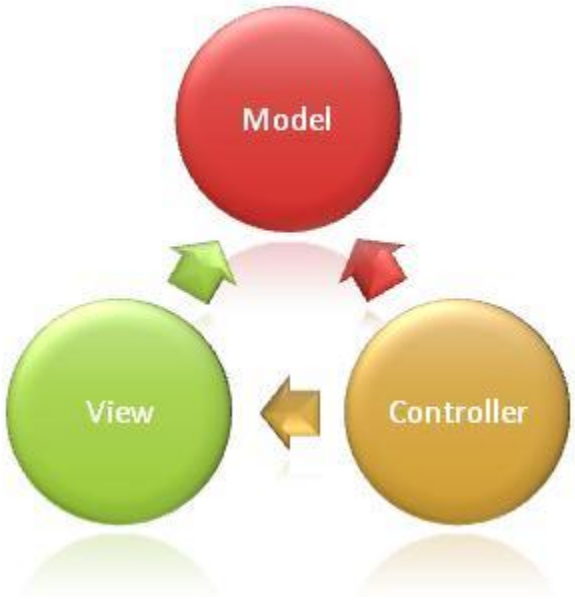
아키텍처 레이어를 나눠보면 Entity Framework 는 Data Access Logic 과 Business Logic 영역에서 역할을 해주고 있다. 소프트웨어 개발 시 어떻게 기준을 잡는가에 따라서 달라질 수 있을 듯 하다. 오늘 알아 볼 내용은 ASP.NET MVC3 를 이용해서 User Presentation 영역과 Entity Framework 상호 관계에 대해서 알아보려 한다.



[그림 9-1] 레이어 구성

<ASP.NET MVC3>

올해 초 ASP.NET MVC3 가 정식으로 릴리즈되었다. MVC 는 익히 아시다시피 Model-View-Controller 패턴으로 각각의 영역이 나눠서 독립적으로 작업이 수행되며 서로간의 관계를 통해서 상호보완적으로 구성되는 설계 방법론이라고 볼 수 있다. MVC 에 대해서는 ASP.NET 파트에서 상세히 설명이 되고 있기에 간략히 언급만 하겠다.



[그림 9-2] MVC 패턴

ASP.NET MVC 3 RTM 다운로드 URL

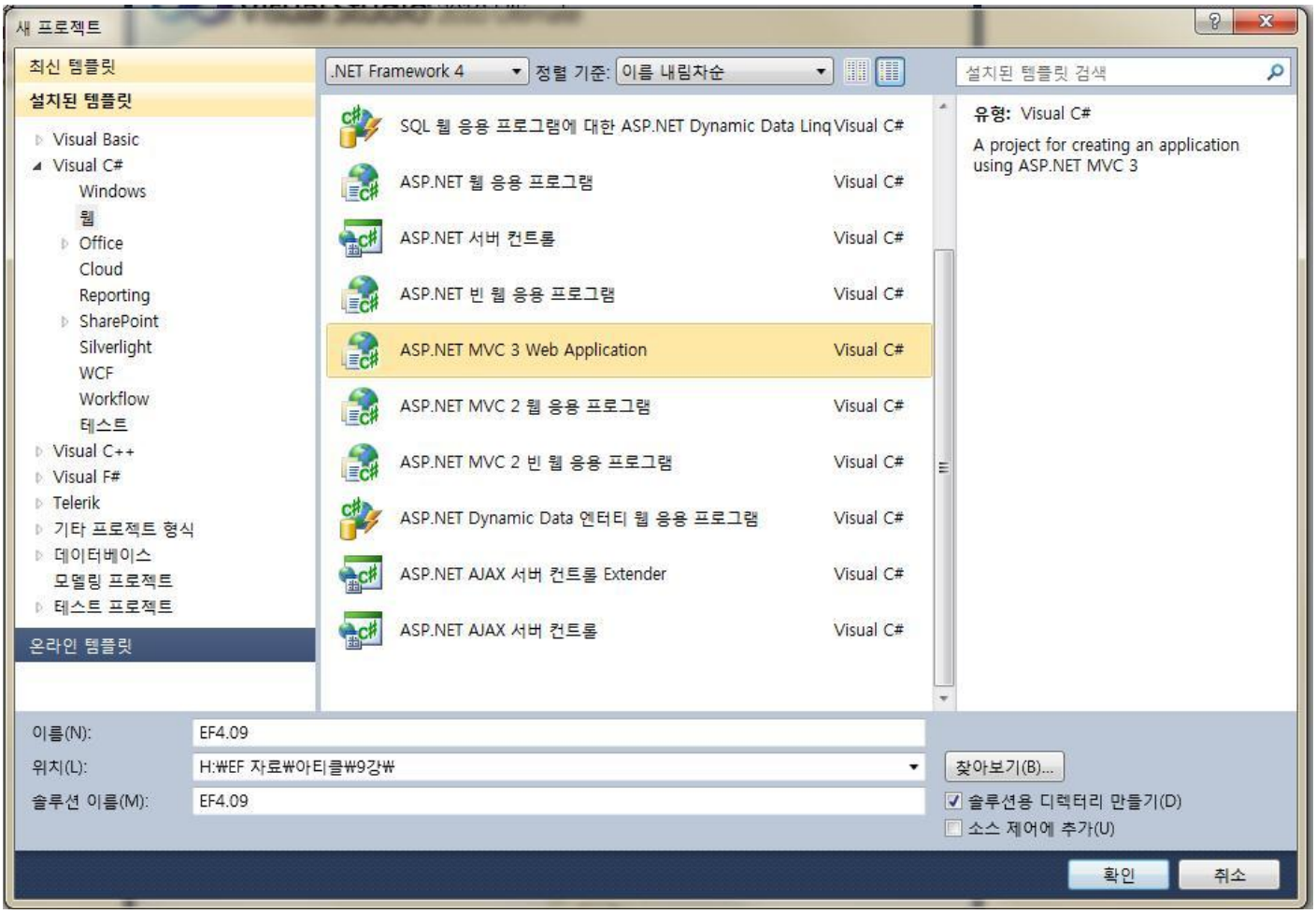
http://www.microsoft.com/download/en/details.aspx?id=4211

ASP.NET MVC 3 RTM Tools Update

http://www.microsoft.com/downloads/ko-kr/details.aspx?familyid=82cbd599-d29a-43e3-b78b-0f863d22811a&displaylang=ko

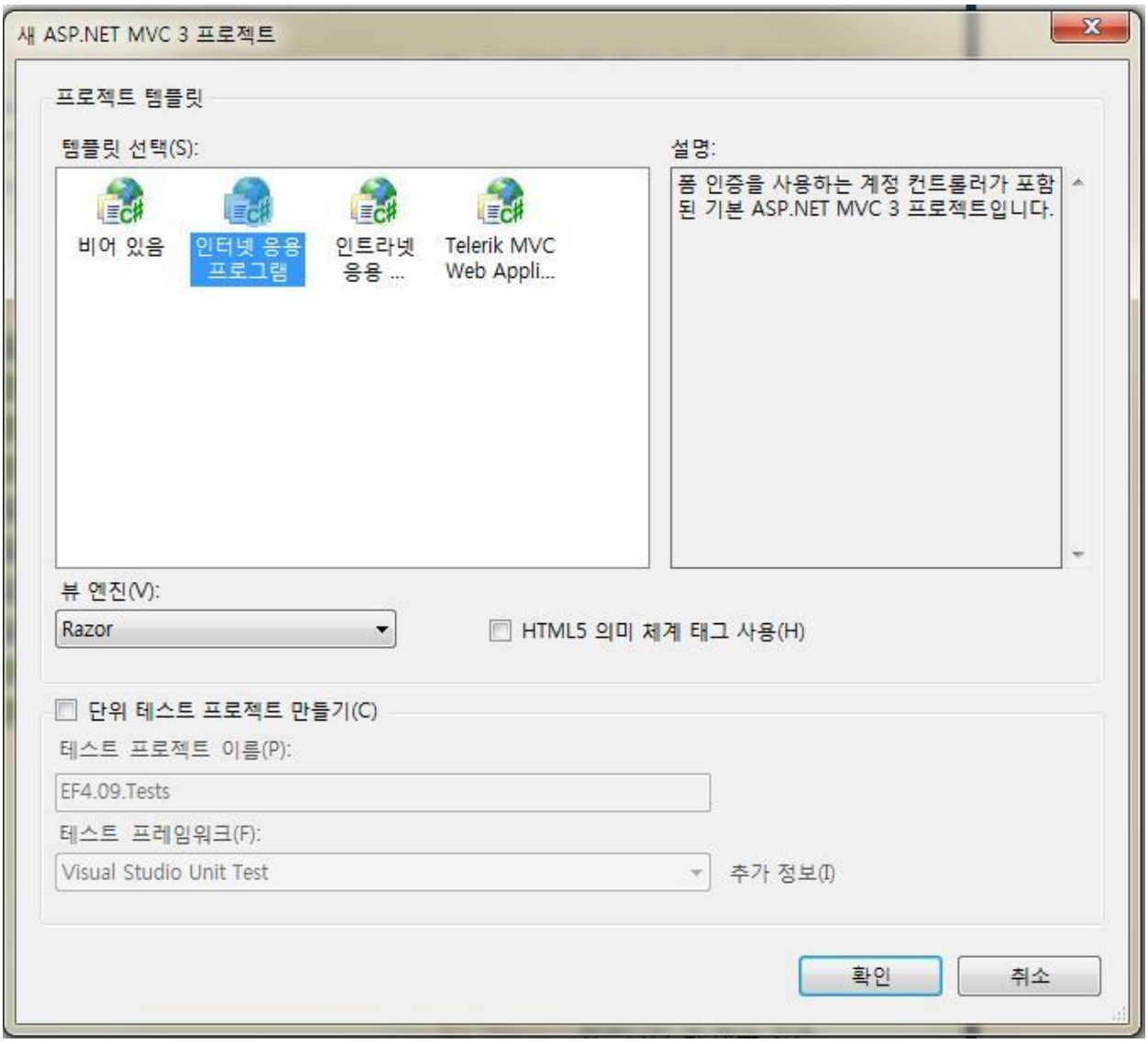
<프로젝트 생성 및 구성>

ASP.NET MVC With Razor 로 웹 프로젝트를 생성한다. 새 프로젝트에서 다음 항목을 선택해준다.



[그림 9-3] ASP.NET MVC 웹 프로젝트 추가

MVC3 는 Razor 엔진을 제공하고 있다. Razor 로 작업을 할 경우 보다 빠른 생산성을 보여줍니다. 개인적인 생각으론 웹페이지가 가벼워지는 느낌도 있다.

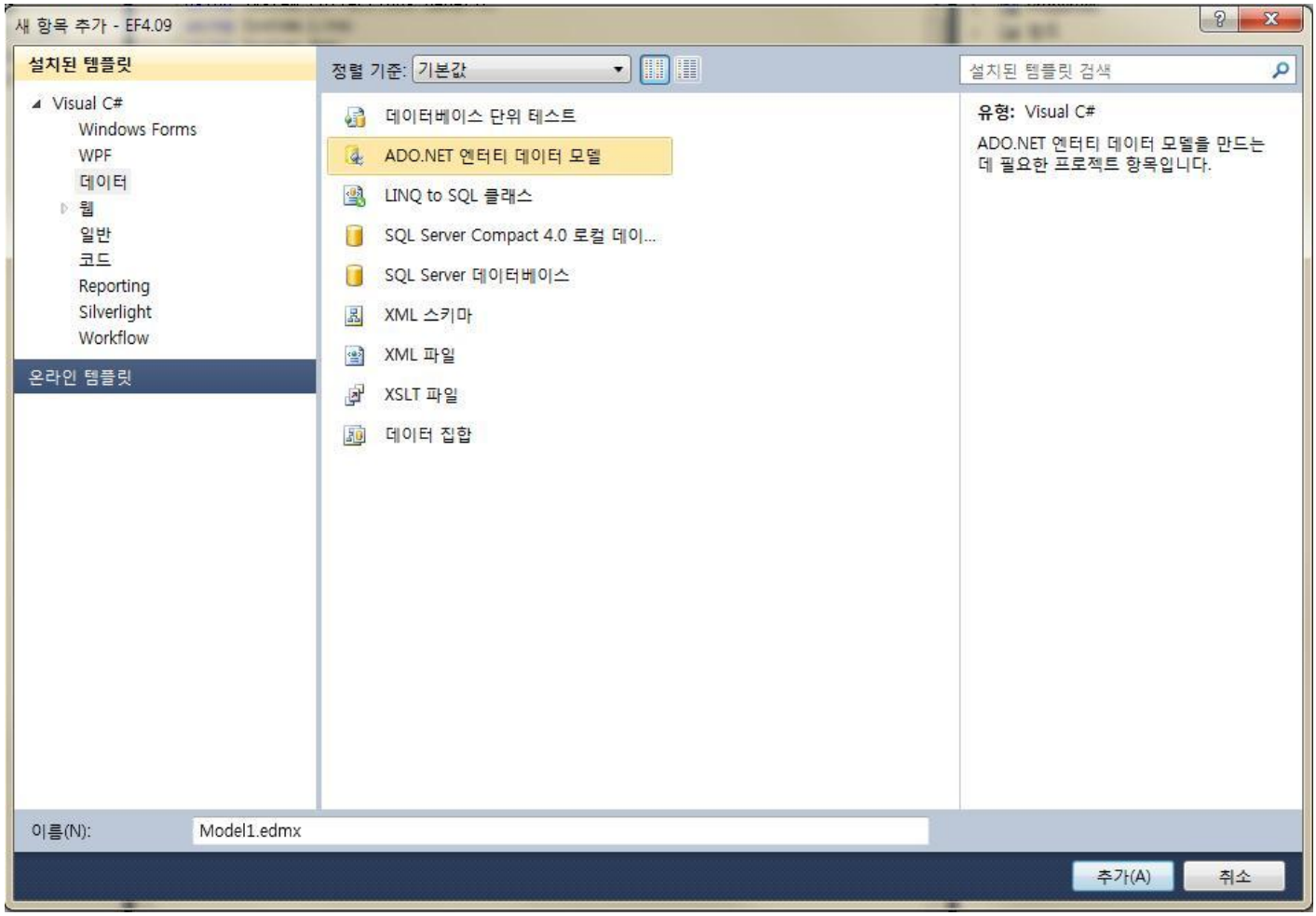


[그림 9-4] Razor 선택

웹 프로젝트를 생성하였다면 F5 를 눌러서 실행을 해보겠다. 기본적으로 구성된 레이아웃에 필수 기능들을 탑재(?)된 화면을 보실 수 있다. 저희가 여기에 추가 기능을 구현하게 된다.

[그림 9-5] MVC 기본 레이아웃 화면

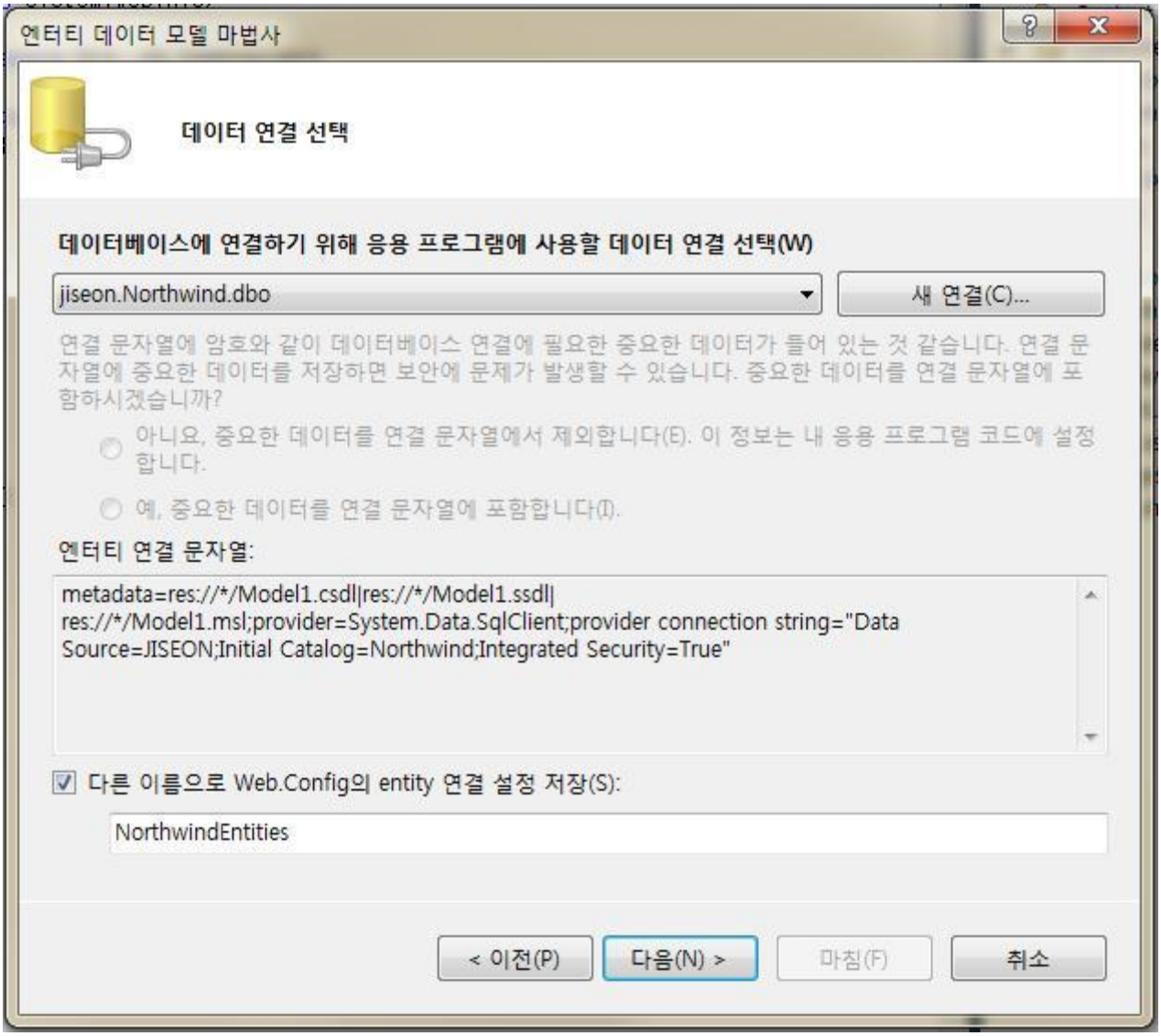
자! 첫 번째 작업은 Entity 를 추가한다.



[그림 9-6] ADO.NET 엔티티 데이터 모델 항목 추가

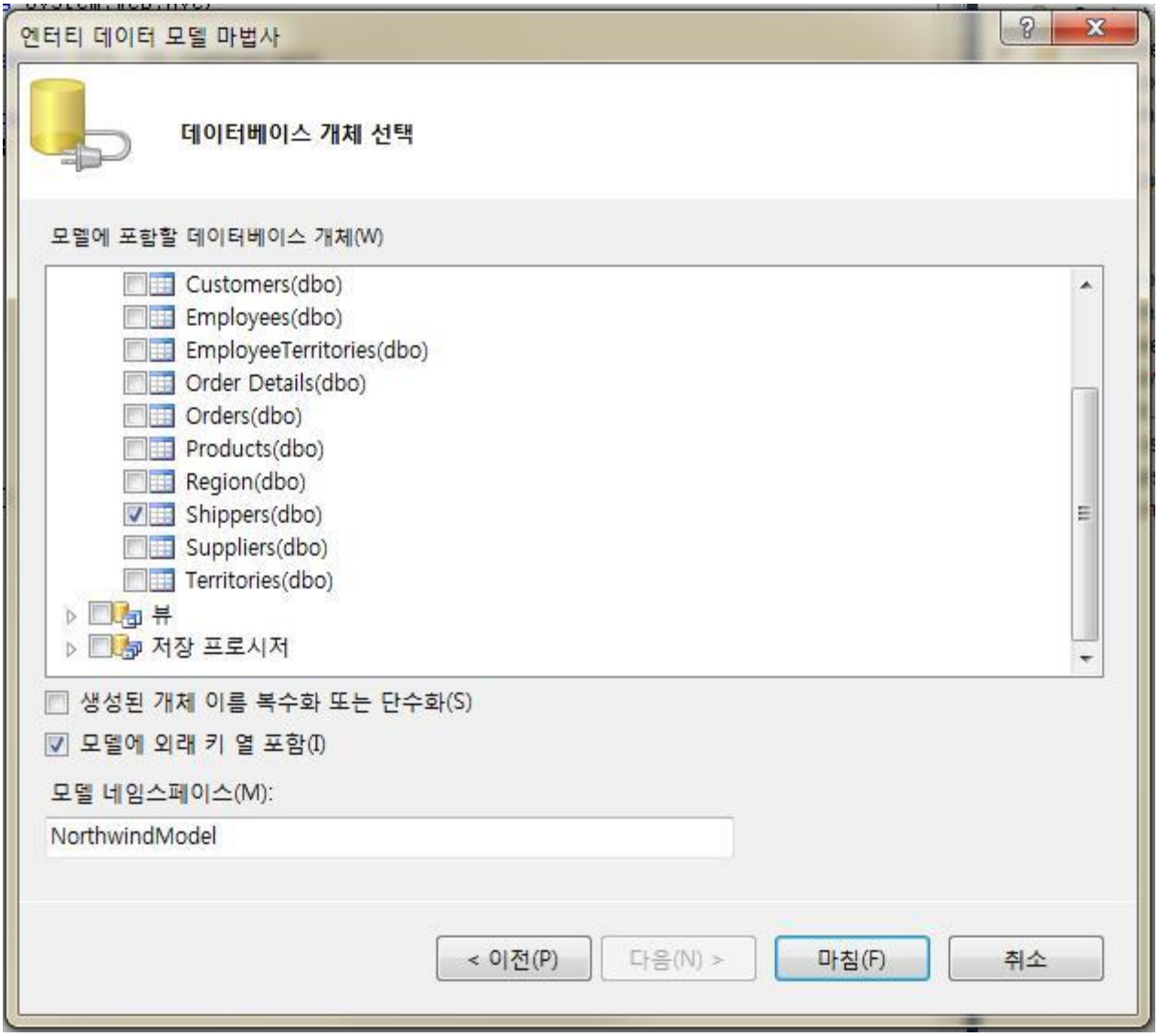
계속적으로 반복되는 "Northwind" 데이터베이스를 연결해준다. 아! 참고로 Northsind 와 pubs 기본 데이터베이스는 다음 URL 에서 다운 가능 하다.

http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=23654



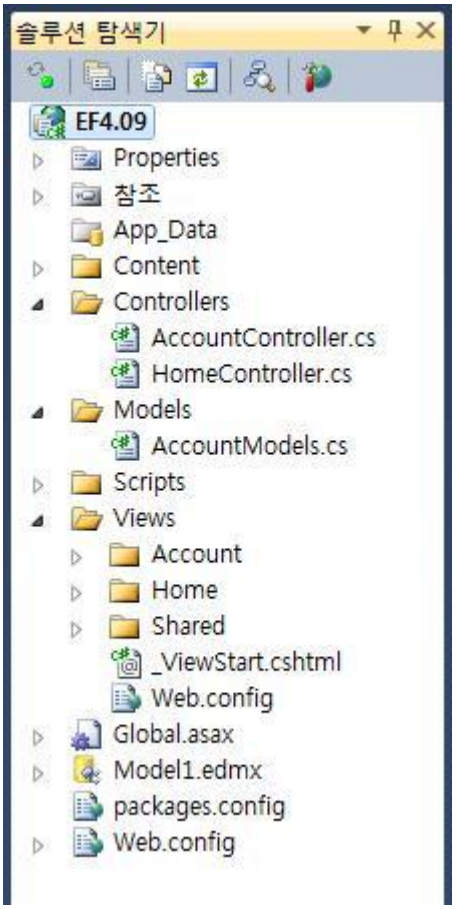
[그림 9-7] DBMS 연결

개체는 “Shippers”테이블만 선택해 준다.



[그림 9-8] 엔티티 선택

추가를 다하셨다면 프로젝트는 다음과 같이 구성이 되었다.



[그림 9-9] 프로젝트 구성

이제 프로젝트 구성이 완료되었다. 환경이 셋팅이 안되신 분들은 설정하느라 시간을 좀 투자하셨을 듯 하다. 이번 세션은 환경설정 및 프로젝트 구성하는 것으로 마무리 하겠다.

[Entity Framework 강좌] 10. Entity Framework – ASP.NET MVC(2)

Entity Framework – ASP.NET MVC(2)

ASP.NET MVC 두 번째 이야기! 저희가 가져온 엔티티를 화면에 구성하는 작업을 진행하겠다. MVC 를 다뤄보신 분도 계시겠지만 처음 접하는 분을 위해 최대한 쉽게 설명 드리겠다.

<ASP.NET MVC Razor>

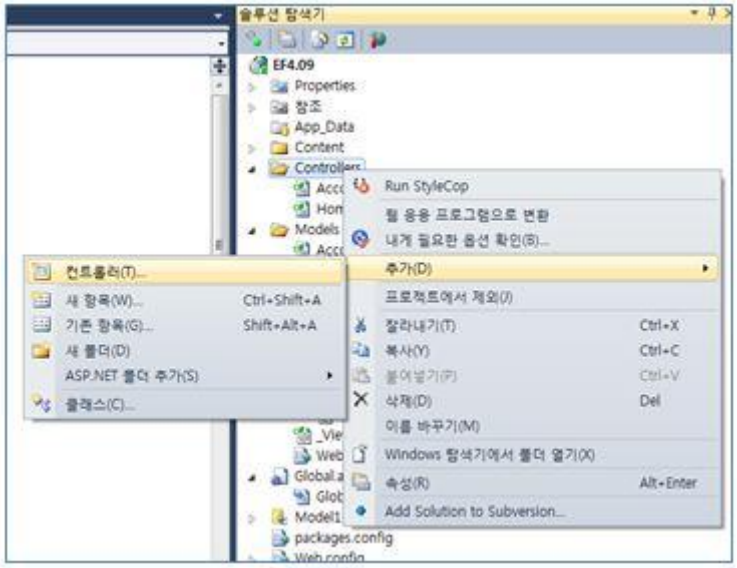
[그림 10-1] Razor 기본 구성 화면

첫 화면 설정은 Global.asax 페이지에서 설정하게 된다.

controller 명과 action 명을 지정하면 "http://localhost:49999/Home/Index"처럼 URL 이 구성된다.

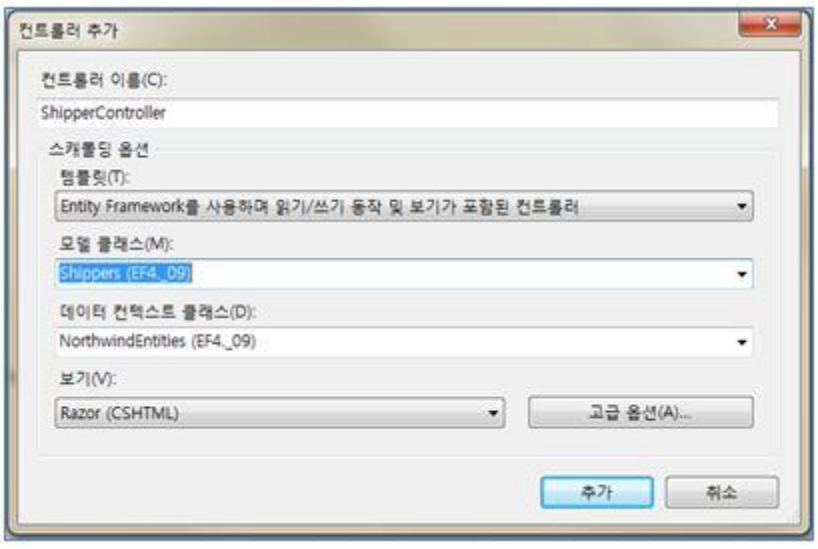
```
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // 경로 이름
                "{controller}/{action}/{id}", // 매개 변수가 있는 URL
                new { controller = "Home", action = "Index", id = UrlParameter.Optional } // 매개
변수 기본값
            );
        }
```

Controllers 폴더에서 컨트롤러를 추가해보겠다. 다음 화면처럼 메뉴를 클릭해준다.



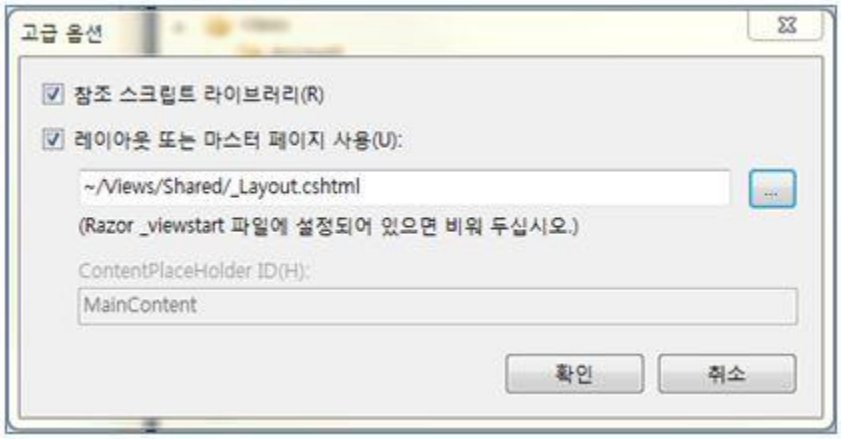
[그림 10-2] 컨트롤러 추가 메뉴

컨트롤러명을 작성하고 스캐폴딩 옵션에 다음과 같이 설정해준다. 모델 클래스는 저희가 추가한 "Shippers"를 지정하면 된다. 데이터 컨텍스트 클래스는 역시 저희가 생성한 "NorthwindEntitys"를 지정한다.



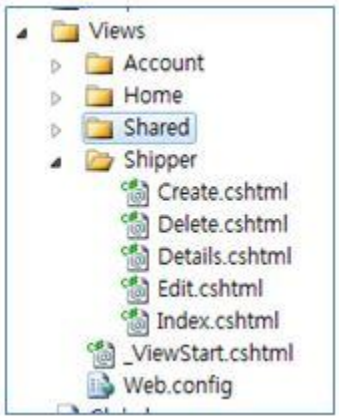
[그림 10-3] 컨트롤러 추가

고급 옵션에서 레이아웃을 설정해주시면 메인화면 레이아웃을 그대로 구성된 페이지를 만날 수 있다.



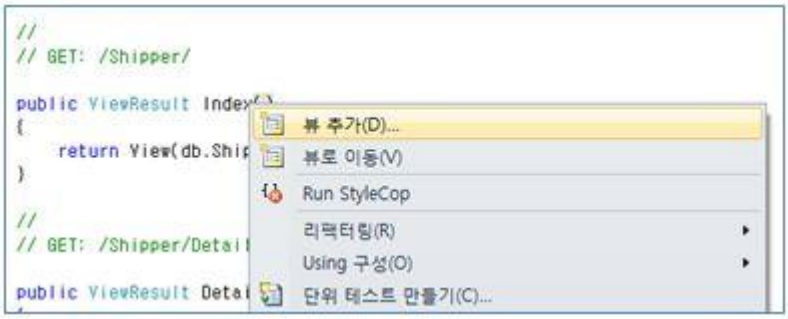
[그림 10-4] 고급 옵션 설정

생성하면 ShipperController 클래스에 많은 메서드들이 자동으로 생성된 것을 볼 수 있다. 또한 스캐폴딩 옵션으로/Views/Shipper 폴더에 View 페이지가 자동으로 생성된 것을 확인 할 수 있다.



[그림 10-5] 뷰 구성

참고로!!! 자동생성하지 않고 뷰를 추가하고 싶다면 컨트롤러에서 생성하려는 메서드에 마우스를 클릭해서 View(화면)을 생성해주면 된다.



[그림 10-6] 뷰 추가 메뉴

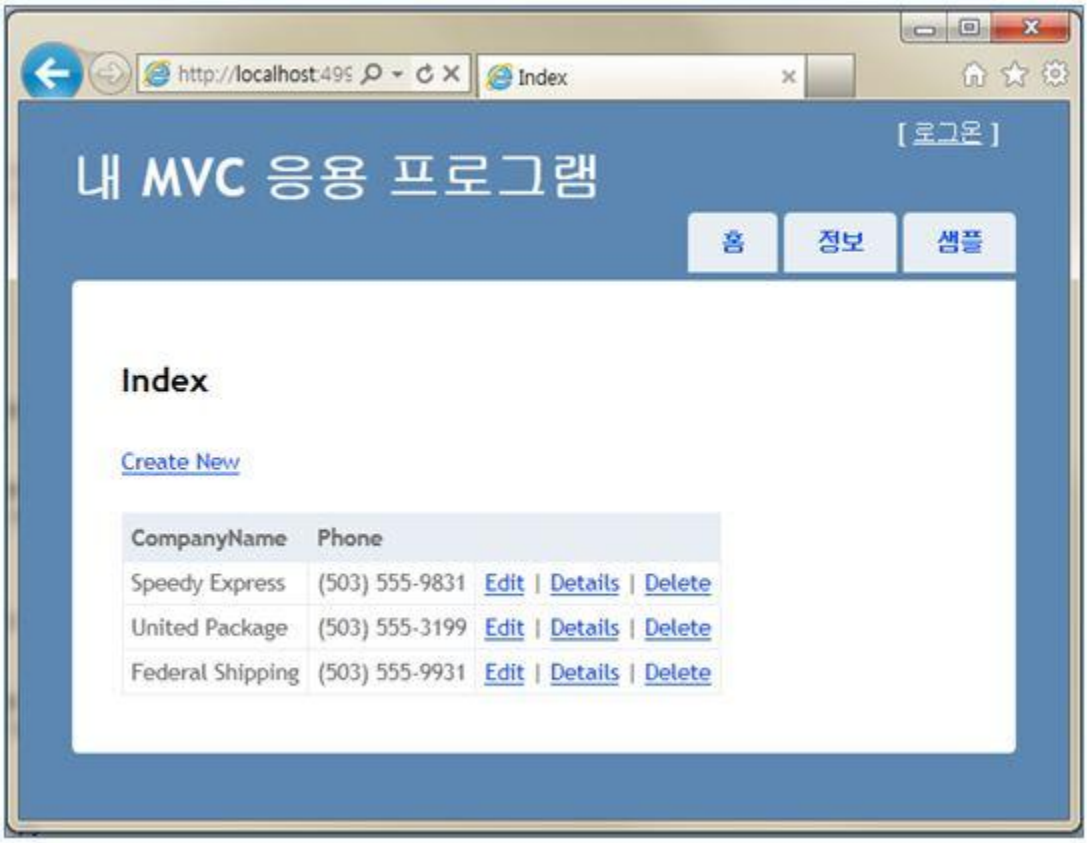
공통 레이아웃에서 "Shipper"의 뷰로 가는 링크를 추가해보겠다.

[그림 10-7] 레이아웃 페이지

메뉴 영역에 샘플 링크를 다음과 같이 추가해준다.

```
<div id="menucontainer">
    <ul id="menu">
        <li>@Html.ActionLink("홈", "Index", "Home")</li>
        <li>@Html.ActionLink("정보", "About", "Home")</li>
        <li>@Html.ActionLink("샘플", "Index", "Shipper")</li>
    </ul>
</div>
```

F5 를 클릭해서 실행해보겠다. 저희가 추가한 샘플 링크를 클릭하면 다음과 같이 "Shippers" 데이터가 목록을 출력되는 것을 확인 할 수 있다.



[그림 10-8] 샘플 리스트 화면

이번 세션에서는 모델-컨트롤러-뷰에 대해서 생성하여 실행하는 것을 간단히 확인해보았다.

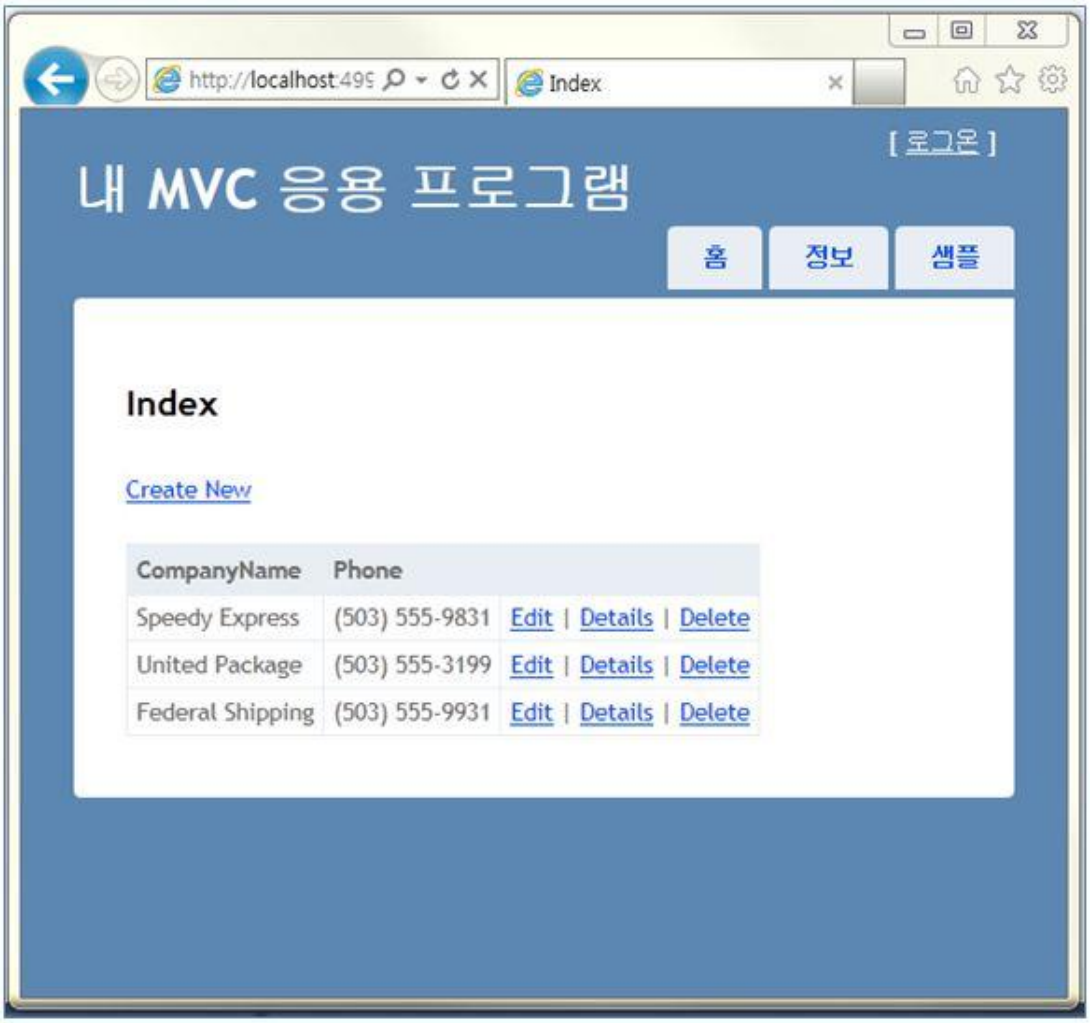
[Entity Framework 강좌] 11. Entity Framework – ASP.NET MVC(3)

Entity Framework – ASP.NET MVC(3)

이번에는 비지니스 로직 영역과 프로세스를 구체적으로 살펴보겠다.

<List 화면>

Shippers 데이터가 목록에 바인딩된 것을 확인 할 수 있다.



[그림 11-1] 리스트 화면

"NorthwindEntities" 컨텍스트의 Shippers 엔티티를 View 에 반환하는 것을 확인 할 수 있다. 화면에서는 뷰 객체를 받아서 처리하여 목록을 출력하고 있다. 구문이 많이 익숙하지 않나요? 앞에서 저희가 엔티티 작업에 대해서 다뤘던 구문과 거의 비슷하다.

```
private NorthwindEntities db = new NorthwindEntities();

public ViewResult Index()
{
    return View(db.Shippers.ToList());
}
```

<상세 화면>

리스트에서 항목을 클릭하면 상세 화면이 다음과 같이 구성된다.

[그림 11-2] 상세 화면

코드를 보면 인자로 id 값을 받고 있다. 유니크한 ShipperID 의 값이다. 컨트롤러에서는 id 값에 해당하는 데이터를 뷰에 반환하고 있다. 뷰에서는 반환된 데이터를 위 화면처럼 뿌려주고 있다.

```
public ViewResult Details(int id)
{
    Shippers shippers = db.Shippers.Single(s => s.ShipperID == id);
    return View(shippers);
}
```

<수정 화면>

다음은 수정 화면이다. 실제로 데이터 변경이 발생하게 된다.

[그림 11-3] 수정 화면

데이터가 변경되거나 삭제 등 비지니스 로직이 추가 될 경우에는 SaveChanges 메서드를 꼭 호출해야 데이터베이스에 반영된다.

```
[HttpPost]
public ActionResult Edit(Shippers shippers)
{
    if (ModelState.IsValid)
    {
        db.Shippers.Attach(shippers);
        db.ObjectStateManager.ChangeObjectState(shippers, EntityState.Modified);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(shippers);
}
```

<신규생성 화면>

다음은 데이터를 생성하는 화면이다. ShipperID 는 자동생성이기에 입력이 안되고 CompanyName 과 Phone 값만 입력을 하면 된다.

[그림 11-4] 신규생성 화면

코드를 보면 좀 다른 것을 확인 할 수 있다. 항목들을 인자로 받는 게 아니라 Shippers 객체 자체를 인자로 받고 있다. 그게 가능한 이유는 뷰의 코드를 확인하면 궁금증이 풀립니다.

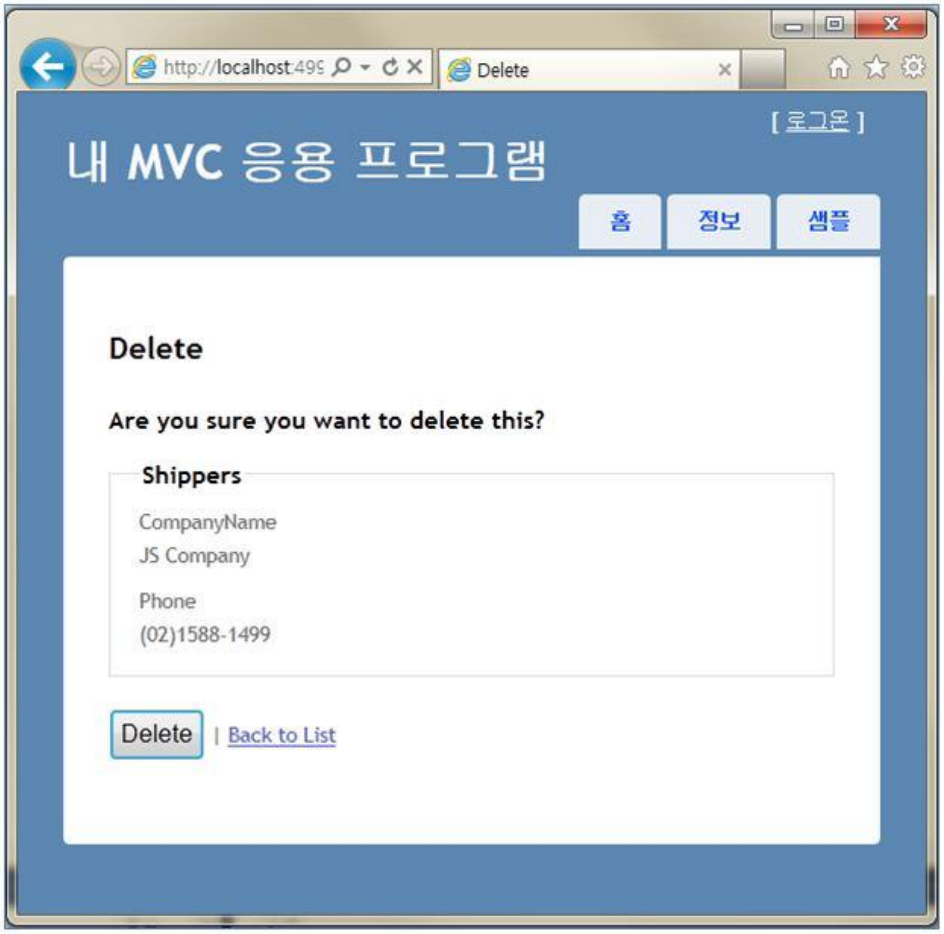
```
[HttpPost]
public ActionResult Create(Shippers shippers)
{
    if (ModelState.IsValid)
    {
        db.Shippers.AddObject(shippers);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(shippers);
}
```

뷰를 구성할 때 모델의 객체를 설정해줬다. 하여 바로 뷰에서 Shippers 객체를 인자로 넘겨 줄 수 있었던 것이다. 만약 뷰에 모델을 할당하지 않는다면 항목들을 하나하나 넘겨서 컨트롤러에서 다시 Shippers 객체에 할당해주는 코드를 작성해야 한다. 하지만 그렇게 작업할 이유는 없다. 바로 모델 객체를 뷰와 컨트롤러 사이의 매개변수로 사용이 가능하기 때문이다.

```
<div class="editor-label">
    @Html.LabelFor(model => model.CompanyName)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.CompanyName)
    @Html.ValidationMessageFor(model => model.CompanyName)
</div>

<div class="editor-label">
    @Html.LabelFor(model => model.Phone)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Phone)
    @Html.ValidationMessageFor(model => model.Phone)
</div>
```

<삭제 화면>

마지막으로 삭제화면이다. 리스트에서 삭제 링크를 클릭하면 확인을 한번 더 진행한다. 이는 프로세스를 어떻게 구성하느냐에 따라 달라집니다. 우선 스캐폴딩 옵션을 이용한 CRUD 화면을 구성하면 기본으로 구성되는 프로세스이다.



[그림 11-5] 삭제 확인 화면

코드는 다음과 같다. 이 또한 SaveChanges 를 최종적으로 호출하고 있다.

```
[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
{
    Shippers shippers = db.Shippers.Single(s => s.ShipperID == id);
    db.Shippers.DeleteObject(shippers);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

MVC 에 JQuery 를 활용한다면 더 세련되고 화려한 웹페이지 구성을 할 수 있다. ASP.NET MVC 와 JQuery 는 각자 파트에서 더 상세히 공부하실 수 있다.

[MyBatis 강좌] 12. MyBatis.Net 들어가기

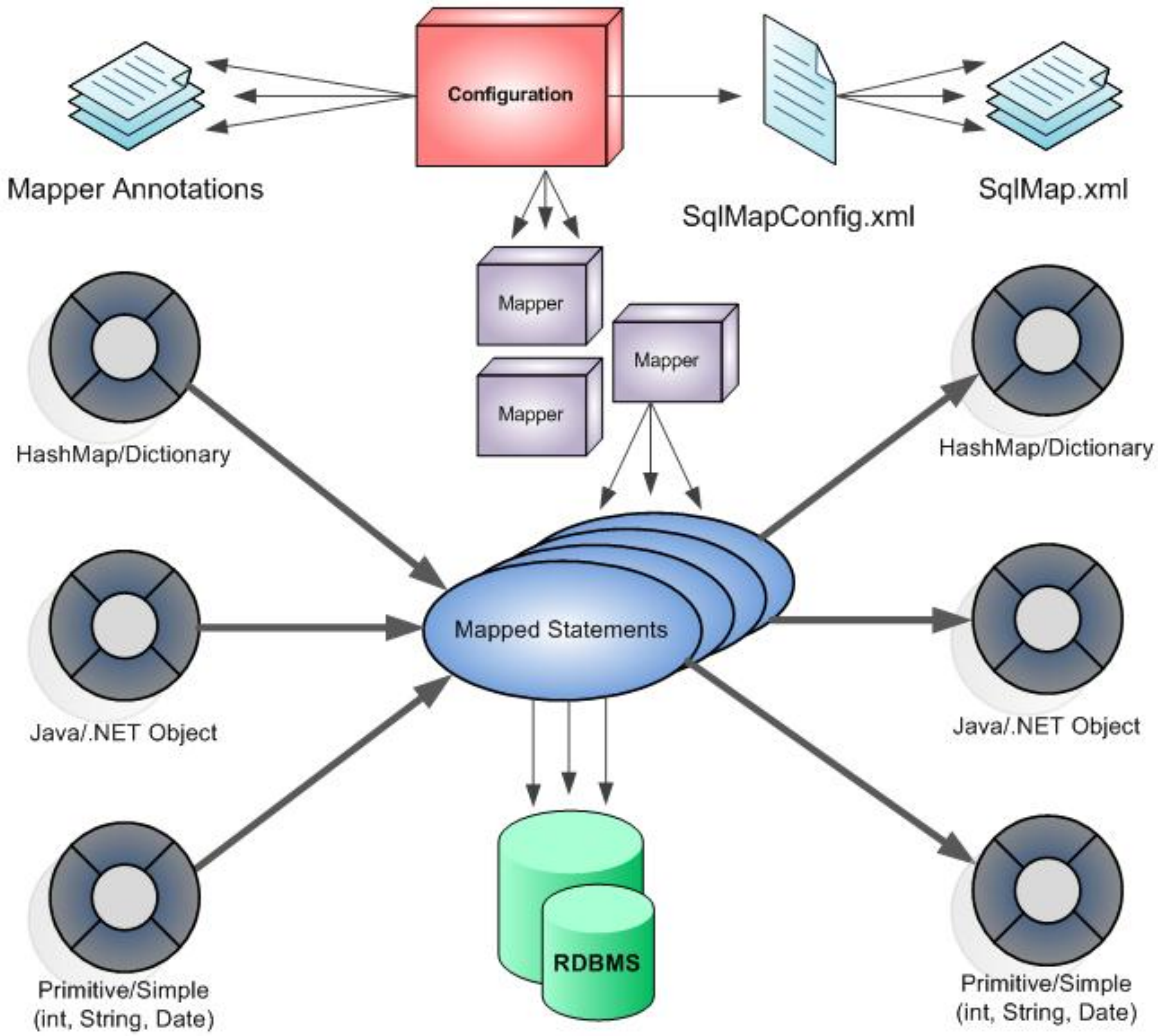
MyBatis.Net 들어가기

이번 세션은 주제는 오픈소스에 대표주자 iBatis 이다. 자바진영에서는 이미 많은 프로젝트에서 iBatis 를 사용하고 있다. 닷넷진영에서도 오픈소스를 프로젝트에 사용하려는 노력이 많이 보인다.
오픈 소스인 iBatis 가 최근에 구글 코드(http://code.google.com)로 옮기면서 MyBatis 로 명칭을 변경하였네요. MyBatis.Net 은 저희가 알고 있는 iBatis 이다.

<MyBatis.NET 소개>

MyBatis.NET 은 저희가 알고 있는 ibatis .NET 버전으로 Data Mapper 프레임워크이다. 최근에 구글 코드로 소스를 옮기면서 mybatis 로 명칭을 변경하였다.
Mybatis 사이트 http://www.mybatis.org 다.

MyBatis.NET 은 객체 지향 응용프로그램과 관계형 데이터베이스에서 데이터 매퍼를 쉽게 제공함으로 응용프로그램에서 코드량을 줄이고 xml 을 쿼리를 작성함으로 관리 측면에서 용이성을 제공하고 있다.
MyBatis.NET 은 흔히 ORM 이라고 생각하지만 결코 ORM 은 아니라고 한다. 테이블 하나하나 객체화하는게 아닌 쿼리(SQL)를 매핑하여 데이터 결과를 반환하게 되기 때문에 Data Mapper Framework 라고 불린다. 객체지향언어의 대표적인 JAVA 와 .NET 을 지원하고 있다.
MyBatis.NET 의 구성도는 다음과 같다.



[그림 12-1] MyBatis.NET

myBatis 는 쿼리(SQL)을 매핑함으로 데이터베이스와 객체 사이에 제약이 따르지 않는다는 점에서도 굉장한 메리트를 가지고 있다고 볼 수 있다.

<MyBatis.NET 관련 소스>

myBatis.net 관련된 소스링크다. 다음 세션부터 실전에 들어가기 전에 다운받아 준다.

.NET Google Code Project: http://code.google.com/p/mybatisnet/

mybatis.net data mapper 다운로드

- Download the Data Mapper for .NET
- Download the Data Mapper User Guide (English)

mybatis.net data access framework 다운로드

- Download the Data Access Framework
- Download the Data Access Framework User Guide (English)

[MyBatis 강좌] 13. MyBatis.NET 기본 및 환경 설정

MyBatis.NET 기본 및 환경 설정

지난 세션에서 MyBatis.NET 에 대해서 기본 그림을 확인 보았다. 아직 MyBatis 라는 단어가 입에 붙지는 않네요. iBatis 라는 단어가 익숙해서 그런가봅니다. MyBatis.NET 은 Data Mapper Framework 로 실제 결과 데이터를 매핑해 줍니다. 이번 세션은 기본 환경 설정하는 부분을 알아보겠다.

<MyBatis.NET 는...>

1. 프로그래밍 코드로부터 SQL 코드를 분리한다.

2. 입력 파라메터를 라이브러리 클래스로 전달하고 출력을 한다.

3. 비지니스 로직 클래스로부터 데이터 액세스 클래스를 분리한다.

4. 자주 사용되는 데이터를 캐싱한다.

5. 트랜잭션과 스레딩 관리가 가능한다.

<MyBatis.NET 관련 소스>

MyBatis.NET 프로젝트 관련 DLL 은 다음 링크에서 다운 받을 수 있다.

mybatis.net data mapper 다운로드

- Download the Data Mapper for .NET

mybatis.net data access framework 다운로드

- Download the Data Access Framework

<프로젝트 참조 파일>

1. **IBatisNet.Common.dll**

DataAccess 와 DataMapper 클래스에서 공유되는 공용 클래스들이다.

2. IBatisNet.DataMapper.dll

DataMapper 프레임워크로 실제 객체 매핑과 결과값 반환 시 사용되는 객체들이다.

3. IBatisNet. DataAccess.dll

DataAccess 객체 프레임워크로 DAO 작업 시 사용된다.

4. providers.config

MyBatis 에서 지원하는 Database Provider 들을 정의 해놓은 파일이다. 저희는 .NET 에서 자주 사용되는 MS SQL Server 를 사용하기에 "sqlServer2.0"Provider 를 사용하게 된다.

Provider	Provider Description
sqlServer1.0	Microsoft SQL Server 7.0/2000 provider available with .NET Framework 1.0
sqlServer1.1	Microsoft SQL Server 7.0/2000 provider available with .NET Framework 1.1
OleDb1.1	OleDb provider available with .NET Framework 1.1
Odbc1.1	Odbc provider available with .NET Framework 1.1
sqlServer2.0	Microsoft SQL Server 7.0/2000/2005 provider available with .NET Framework 2.0
OleDb2.0	OleDb provider available with .NET Framework 2.0
Odbc2.0	Odbc provider available with .NET Framework 2.0
oracle9.2	Oracle provider V9.2.0.401
oracle10.1	Oracle provider V10.1.0.301
oracleClient1.0	MS Oracle provider V1.0.5 available with .NET Framework 1.1
ByteFx	ByteFx MySQL provider V0.7.6.15073
MySql	MySQL provider V1.0.4.20163
SQLite3	SQLite.NET provider V0.21.1869.3794
Firebird1.7	Firebird SQL .NET provider V1.7.0.33200
PostgreSql0.7	Npgsql provider V0.7.0.0
iDb2.10	IBM DB2 iSeries provider V10.0.0.0

5. SqlMap.config

DataMapper 설정 파일이다.해당 파일에는 연결 대상인 Database 기본 정보나 객체 매핑이 정의된 xml 파일 링크 정보를 정의해주는 파일이다. <sqlMaps> 요소 영역을 저희가 자주 편집하게 될 것 이다.

```
<!-- Relative path from the project root directory using a property variable -->
<sqlMaps>
  <sqlMap resource="${root}Maps/Account.xml"/>
  <sqlMap resource="${root}Maps/Category.xml"/>
  <sqlMap resource="${root}Maps/Product.xml"/>
</sqlMaps>

<!-- Embedded resources using [extendednamespace.]filename, assemblyname -->
<sqlMaps>
  <sqlMap embedded="Maps.Account.xml, MyApp.Data"/>
  <sqlMap embedded="Maps.Category.xml, MyApp.Data"/>
  <sqlMap embedded="Maps.Product.xml, MyApp.Data"/>
</sqlMaps>

<!-- Full URL with a property variable -->
<sqlMaps>
  <sqlMap url="C:/${projectdir}/MyApp/Maps/Account.xml"/>
  <sqlMap url="C:/${projectdir}/MyApp/Maps/Category.xml"/>
  <sqlMap url="C:/${projectdir}/MyApp/Maps/Product.xml"/>
</sqlMaps>
```

[그림 13-1] sqlMap.confog sqlMaps 요소

MyBatis.NET 는 DLL 형태로 제공되기 때문에 환경 설정하는데 어려움이 없다. 프로젝트 실전을 통해서 작업을 해보시면 생각보다 진입하기가 쉽다는 것을 느낄 수 있다. 다음 세션에 작업할 때 위에서 언급된 항목들이 사용 된다. XML 에서 작업이 이뤄지기 때문에 헷갈릴 수도 있지만 오늘 언급한 항목들에 대한 인지만 가지고 있으면 쉽게 따라 올 수 있다.

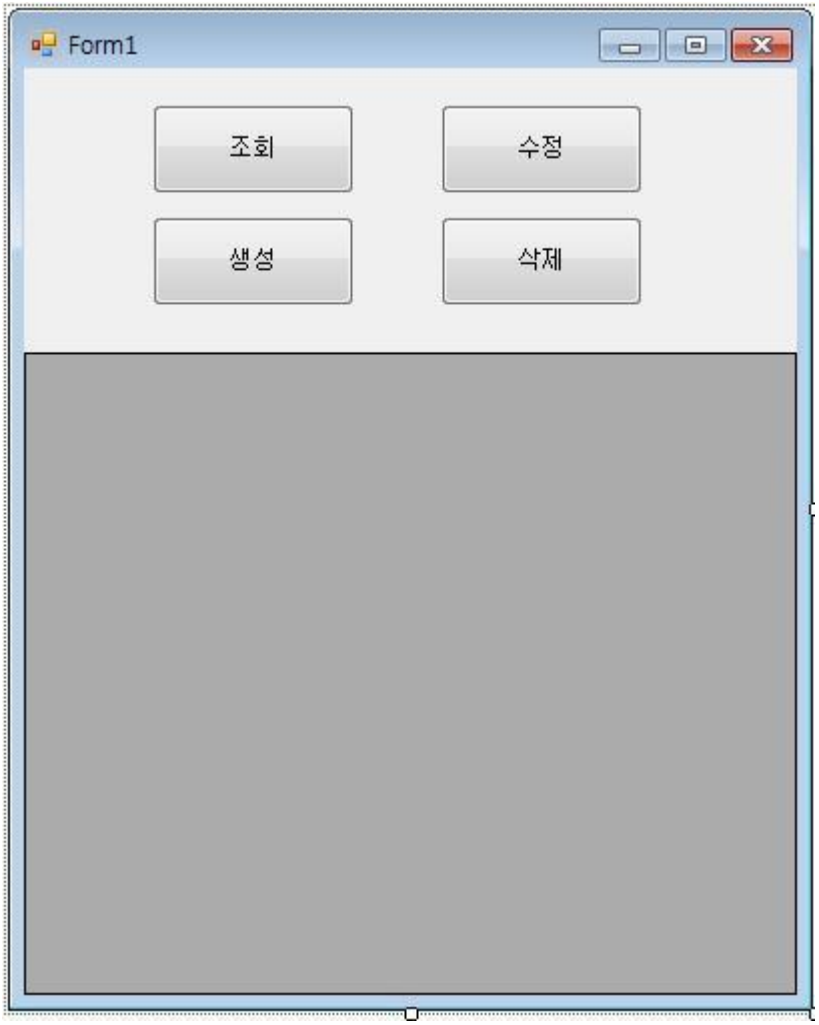
[MyBatis 강좌] 14. MyBatis.NET CRUD(1)

MyBatis.NET CRUD(1)

MyBatis.NET 의 아주 기본적인 작업에 대해서 살펴보려 한다. 프로젝트를 생성하고 데이터를 조회하는데 객체 매핑을 어떻게 하는지 확인 하실 수 있다.

<프로젝트 생성>

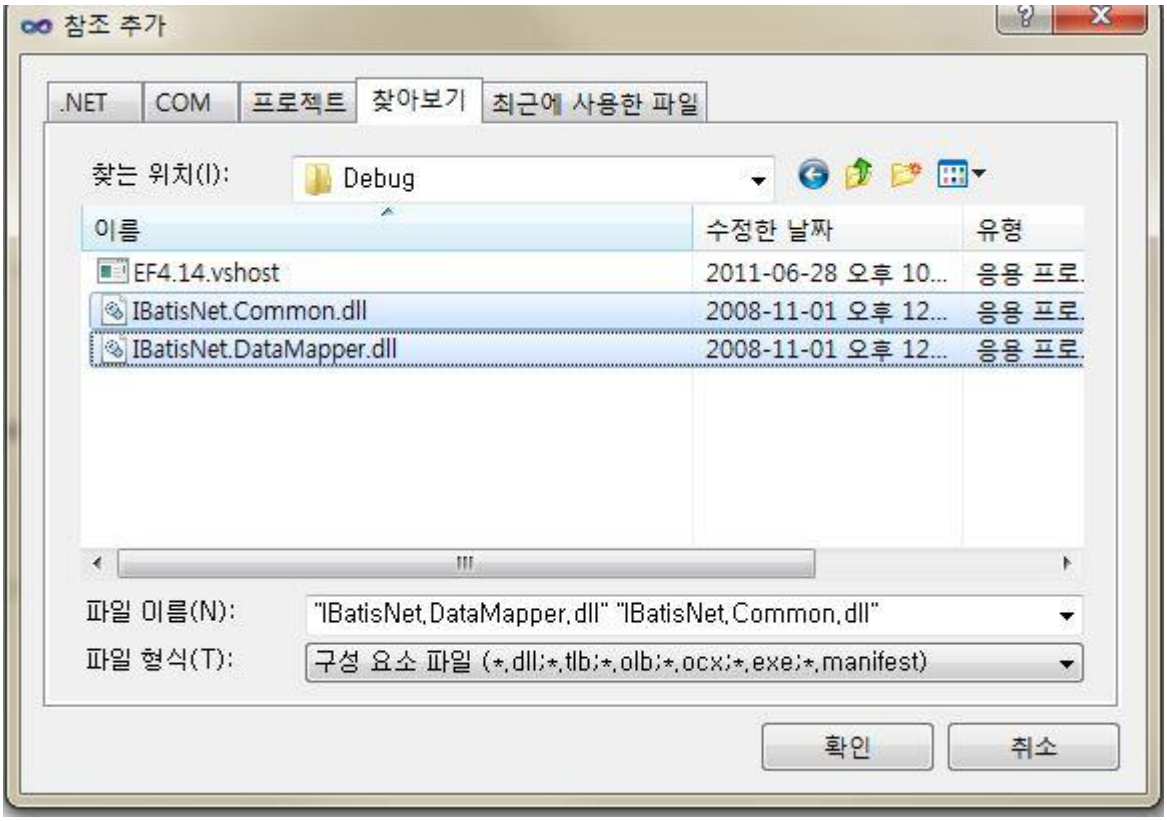
우선 간단한 CRUD 를 확인하는 작업이기에 응용프로그램을 다음과 같이 디자인 해보겠다.



[그림 14-1] UI 구성

<초기 작업>

Data Mapper 작업을 위해서 초기에 설정해줘야 하는 작업들이 있다. 앞 세션에서 관련된 파일들을 언급했다. 해당 파일들이 어떤 역할들을 하는지 살펴보겠다. MyBatis.Net 사이트에서 다운 받은 파일 중에 IBatisNet.Common.dll 과 IBatisNet.DataMapper.dll 을 해당 프로젝트에 참조 해준다.

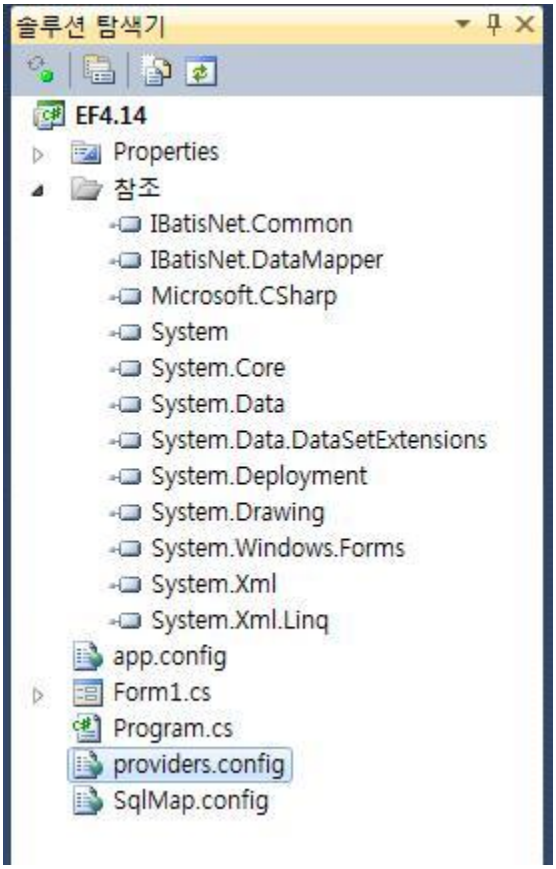


[그림 14-2] dll 참조

₩IBatis.DataMapper.1.6.2.bin₩Ibatis.DataMapper.1.6.2.bin 폴더에 아래 config 파일이 존재한다. 프로젝트에 추가해준다. 새로 만들어도 되지만 xml 로 작성되기 때문에 오류율을 줄이기 위해 가져다 편집을 하는 방법을 선택하겠다.

[그림 14-3] config 파일 참조

여기까지 진행하셨다면 다음과 같이 프로젝트가 구성된다.



[그림 14-4] 프로젝트 구성

<Config 파일 설정>

providers.config 파일 내용을 확인하면 많은 DBMS Provider 가 정의되어 있다. 저희가 사용하는 Provider 는"sqlServer2.0"이다. 다른 Provider 는 enabled="false" 해주시고 "sqlServer2.0"는 는 enabled="true"로 설정해준다.

```
<provider name="OleDb1.1"
          description="OleDb, provider V1.0.5000.0 in framework .NET V1.1"
          enabled="true"
          assemblyName="System.Data, Version=1.0.5000.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
    connectionClass="System.Data.OleDb.OleDbConnection"
          commandClass="System.Data.OleDb.OleDbCommand"
          parameterClass="System.Data.OleDb.OleDbParameter"
          parameterDbTypeClass="System.Data.OleDb.OleDbType"
          parameterDbTypeProperty="OleDbType"
          dataAdapterClass="System.Data.OleDb.OleDbDataAdapter"
          commandBuilderClass="System.Data.OleDb.OleDbCommandBuilder"
          usePositionalParameters="true"
          useParameterPrefixInSql="false"
          useParameterPrefixInParameter="false"
          parameterPrefix=""
    allowMARS="false"
    />
```

다음은 SqlMap.config 파일이다. 데이터베이스 연결 영역에 다음과 같이 DB 연결설정을 해준다.

```
        <!-- Database connection information -->
        <database>
              <provider name="sqlServer2.0"/>
              <dataSource name="Northwind" connectionString="Data Source=JISEON;Initial
Catalog=Northwind;Integrated Security=True;"/>
        </database>
```

<데이터 조회 >

기본 설정이 마무리 되었으니 이제부터 데이터를 한번 조회해보겠다. MyBatis 는 쿼리와 결과 데이터의 매퍼 역할을 하는 프레임워크다. 그래서 단순히 매퍼 역할만 하게 된다. 그 부분을 xml 로 정의한다. Northwind 데이터베이스에서 Shippers 테이블을 조회 한다. 프로젝트에 매퍼 역할을 하는 Shippers.xml 파일을 생성해야 한다. 또한 응용프로그램에서 용이하게 사용하는 Shippers 의 엔티티를 Shippers.cs 로 구성해 보겠다.

Shippers.cs 는 다음과 같이 작성해준다.

```
public class Shippers
{
    public int ShipperID { get; set; }
    public string CompanyName { get; set; }
    public string Phone { get; set; }
}
```

다음은 Shippers.xml 내용을 확인해보죠!! Select statement 를 다음과 같이 작성해주면 된다.

```
<?xml version="1.0" encoding="utf-8" ?>

<sqlMap namespace="EF4._14" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://ibatis.apache.org/mapping">

    <alias>
        <typeAlias alias="Shippers"  type="EF4._14.Shippers" />
    </alias>

    <statements>
        <select id="SelectShippers"  resultClass="Shippers">
            select * from Shippers
        </select>
    </statements>

</sqlMap>
```

UI 에 다음 네임스페이스를 추가한다.

```
using IBatisNet.Common;
using IBatisNet.DataMapper;
```

조회 버튼을 클릭해서 데이터를 조회해보겠다. Mapper 객체를 통해서 데이터를 조회하여 결과값을 리턴한다.

```
    private void btnSelect_Click(object sender, EventArgs e)
    {
        try
        {
            IList<Shippers> list = Mapper.Instance().QueryForList<Shippers>("SelectShippers",
null);
            dataGridView1.DataSource = list;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.ToString());
        }
    }
```

조회된 화면이다. 조회를 한 번하기 위해 긴 여정을 걸쳤다. 그래도 MyBatis 를 사용하는 장점이 뭔가 있겠죠?! 다음 편에서 그 진실을 파헤져 보겠다.

[그림 14-5] 조회 결과 화면

Tip!

- 배포 시 Bin 폴더에 config 파일과 xml 파일이 같이 반영되어야 한다.

[MyBatis 강좌] 15. MyBatis.NET CRUD(2)

MyBatis.NET CRUD(2)

MyBatis.NET Mapper 를 이용한 조회를 앞 세션에서 진행했다. 추가로 신규생성, 수정, 삭제를 진행해 보겠다.

<데이터 수정>

Shippers.xml 에 statement 를 다음과 같이 정의해준다. Shippers 테이블 업데이트 구문이다.

```
<update id="UpdateShippers" parameterClass="Shippers">
        UPDATE [Shippers]
        SET CompanyName = #CompanyName#, Phone = #Phone#
        WHERE ShipperID = #ShipperID#
</update>
```

수정 버튼을 클릭했을때 Mapper 클래스를 이용해서 Update 메서드에 위에서 정의한 UpdateShippers ststement 명을 작성해줍니다. 이렇게 되면 Mapper 클래스는 ststement 명을 찾아 해당 쿼리를 DB 에 던지게 된다.

```
        private void btnUpdate_Click(object sender, EventArgs e)
        {
            try
            {
                Shippers shipper = new Shippers();
                shipper.ShipperID = 4;
                shipper.CompanyName = "SM Company";
                shipper.Phone = "(02)0000-4444";
                Mapper.Instance().Update("UpdateShippers", shipper);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
```

[그림 15-1] 수정 결과 화면

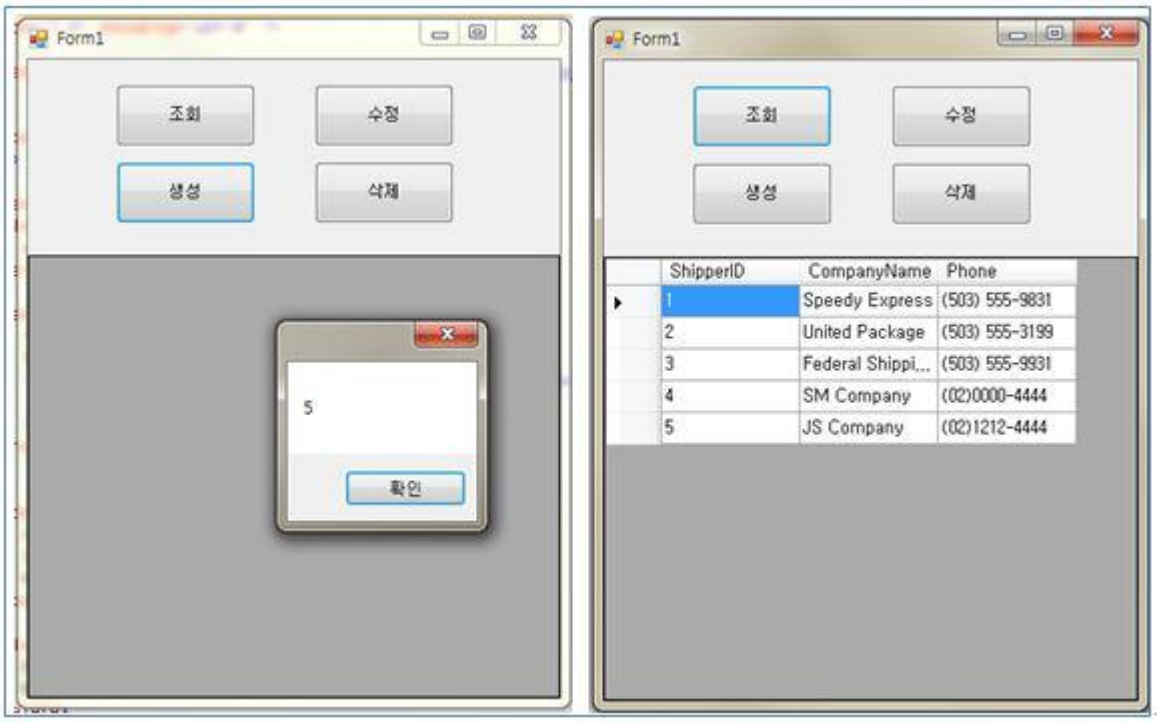
<데이터 신규생성 >

데이터 신규생성 작업이다. Shippers.xml 에 insert statement 를 작성해 주기 바란다.

```
<insert id="InsertShippers" parameterClass="Shippers">
        INSERT INTO [Shippers](CompanyName, Phone)
        VALUES (#CompanyName#, #Phone#)
        <selectKey type="post" resultClass="int" property="ShipperID">
              select @@IDENTITY as value
        </selectKey>
</insert>
```

코드에서 다음과 같이 호출해준다.

```
private void btnInsert_Click(object sender, EventArgs e)
{
    try
    {
        Shippers shipper = new Shippers();
        shipper.CompanyName = "JS Company";
        shipper.Phone = "(02)1212-4444";
        object obj = Mapper.Instance().Insert("InsertShippers", shipper);
        MessageBox.Show(obj.ToString());
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```



[그림 15-2] 추가 후 결과 화면

<데이터 삭제>

마지막으로 데이터 삭제이다. Shippers.xml 에 delete statement 가 필요하겠죠? 다음과 같이 작성해준다.

```
<delete id="DeleteShippers" parameterClass="Shippers">
        DELETE FROM [Shippers]
        WHERE ShipperID = #ShipperID#
</delete>
```

코드는 다음과 같이 삭제메서드를 호출해주시면 된다.

```
        private void btnDelete_Click(object sender, EventArgs e)
        {
            try
            {
                Shippers shipper = new Shippers();
                shipper.ShipperID = 4;
                Mapper.Instance().Delete("DeleteShippers", shipper);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
```

[그림 15-3] 삭제 후 결과 화면

<MyBatis.NET 의 장점은?>

ORM 하면 대표적을 생각하는 프레임워크 중에 하나인 MyBatis.NET 실제로 작업을 해보니 xml 파일을 핸들링하는 것과 중간에 Mapper 를 통해서 데이터 결과를 받아오는 등의 작업이 귀찮지 않다고 한다면 거짓말 일 것이다. 그럼에도 불구하고 사랑을 받고 있는 MyBatis.NET 의 장점은 무엇일까? 레거시 시스템 경우 기존 데이터베이스 연동을 해야 하는 경우에 중간 매퍼를 통해서 처리함으로 관리 측면의 용이성이 있다. 또한 파일로 관리되기 때문에 실제 로직 실행 시에 파일을 읽어와서 처리하게 된다. 결국은 코드에 쿼리문이 작성되지 않고 별도로 관리된다는 측면에서 많은 이점을 가질 수 있을듯 하다.

소스코드 : EF4.14.zip

[NHibernate 강좌] 16. NHibernate 들어가기

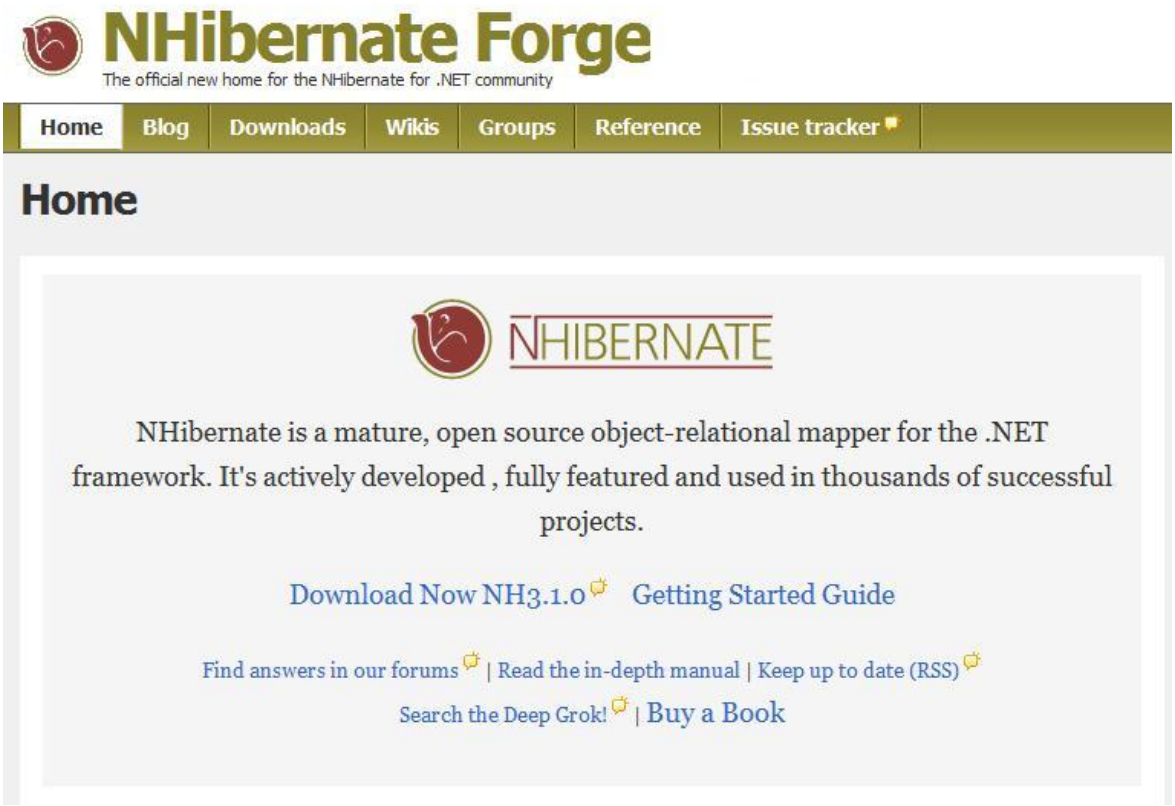
NHibernate 들어가기

오픈소스 ORM 프레임워크에서 손에 꼽히는 Hibernate 프레임워크에 대해서 알아보려 한다. NHibernate 는 Hibernate 닷넷 버전으로 NHibernate 경우 다양한 DBMS 를 지원함으로 많은 프로젝트에서 ORM 프레임워크로 활용 되고 있다.

<NHibernate 소개>

NHibernate 공식 사이트 http://nhforge.org

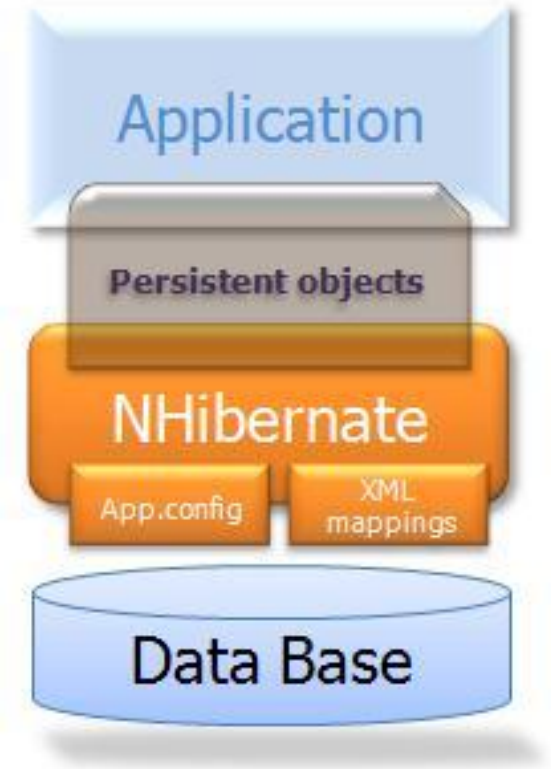
메인 화면에서 관련 파일을 다운로드 받을 수 있다. NHibernate 3.1.0 까지 릴리즈 되었다.



[그림 16-1] NHibernate 공식 사이트

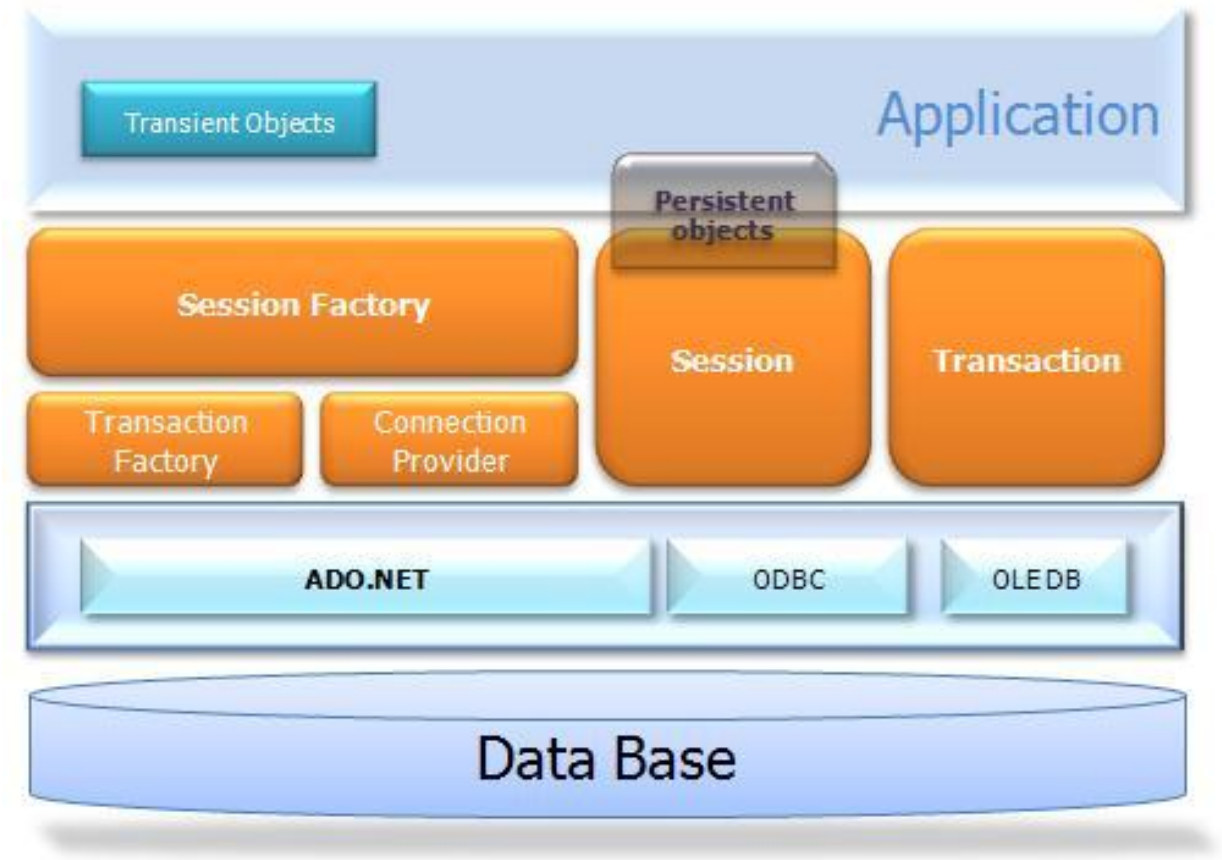
<NHibernate Overview>

NHibernate 는 매우 높은 수준의 아키텍처이다. 응용프로그램과 NHibernate 사이에 영속 객체(Persistent Objects)는 Entity Framework 4.0 에서 추가된 영속성을 지원하기 위한 객체이다. 객체와 데이터베이스 테이블 간의 매핑을 XML 로 관리한다.



[그림 16-2] NHibernate 아키텍처

좀 더 구체적으로 살펴보겠다.



[그림 16-3] NHibernate Layer

- **ISessionFactory (NHibernate.ISessionFactory)**

threadsafe 캐시로 단일 데이터베이스를 사용하는 경우 컴파일된 매핑을 캐시하여 사용 가능하면 ISession 과 IConnectionProvider 의 클라이언트를 위한 factory 이다.

- **ISession (NHibernate.ISession)**

ISession 을 통해서 단일 스레드는 응용프로그램과 영속 저장소 사이에서 존재하게 된다. ADO.NET Connecttion 을 랩핑하고 ITransaction 을 위한 factory 로 보시면 된다.

- **Persistent Objects and Collections**

영속성 상태나 비지니스 함수를 포함하는 단일 스레드 객체로 짧게 살아있는 객체를 담고 있다. 이는 세션이 닫히면 응용프로그램 레이어에서 분리되어 자유롭게 가능하다.

- **Transient Objects and Collections**

ISession 과 관련되지 않는 영속 클래스의 인스턴스를 말한다.

- **ITransaction (NHibernate.ITransaction)**

선택사항이며 ADO.NET transaction 밑에서 사용된다. ISession 는 몇몇 상황에서 ITransactions 을 걸치게 된다.

- **IConnectionProvider (NHibernate.Connection.IConnectionProvider)**

이 또한 선택사항이며 ADO.NET 의 connection 과 command 를 위한 factor 다. 응용프로그램에 노출되지 않지만 개발자에 의해 확장 구현 가능하다.

- **IDriver (NHibernate.Driver.IDriver)**

선택사항이며 ADO.NET provider 사이에서 캡슐화 인터페이스를 제공한다.

- **ITransactionFactory (NHibernate.Transaction.ITransactionFactory)**

선택사항이며 ITransaction 인스턴스를 위한 factory 다. IConnectionProvider 와 마찬가지로 응용프로그램에 노출되지 않지만 개발자에 의해 확장 구현 가능하다.

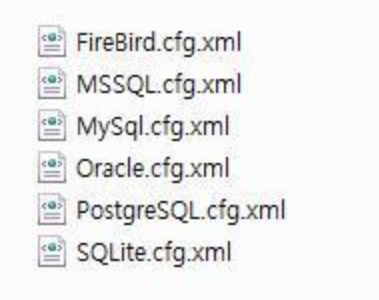
[NHibernate 강좌] 17. NHibernate xml 파일

NHibernate xml 파일

NHibernate 는 ORM 으로 XML 에서 설정을 한다. 오늘은 NHibernate 프로젝트 관련하여 포함된 Config 파일 및 객체 정의하는 xml 파일들을 살펴보고자 한다. 지난 세션에서 소개한 NHibernate 사이트에서 NH3.1.0 다운로드를 받으면 xml 파일 템플릿이 제공되고 있다.

<Configuration_Templates>

다운받은 파일의 ₩NHibernate-3.1.0.GA-bin₩Configuration_Templates 폴더에 Database 설정 관련 xml 이 존재한다.



[그림 17-1] Database 정의 xml

MSSQL.cfg.xml 파일을 열어보자. MS SQL Server 연결관련 정보가 설정되어 있다. 우리는 여기 정보를 프로젝트 app.config 나 web.config 에 정의하게 될 것이다.

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-configuration  xmlns="urn:nhibernate-configuration-2.2" >
        <session-factory name="NHibernate.Test">
                <property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
                <property name="connection.connection_string">
                        Server=(local);initial catalog=nhibernate;Integrated Security=SSPI
                </property>
                <property name="adonet.batch_size">10</property>
                <property name="show_sql">false</property>
                <property name="dialect">NHibernate.Dialect.MsSql2000Dialect</property>
                <property name="use_outer_join">true</property>
                <property name="command_timeout">60</property>
                <property name="query.substitutions">true 1, false 0, yes 'Y', no 'N'</property>
                <property
name="proxyfactory.factory_class">NHibernate.ByteCode.LinFu.ProxyFactoryFactory,
NHibernate.ByteCode.LinFu</property>
        </session-factory>
</hibernate-configuration>
```

<table>.hbm.xml 정의

NHibernate 는 테이블 기준으로 매핑을 한다. 하여 테이블마다 xml 파일이 존재한다고 보시면 된다. 샘플로 하나 살펴보겠다. 구조만 확인을 해준다. 저희가 자주 사용하는 Northwind 데이터베이스 Shippers 테이블을 정의해놓은 파일이다. **Shippers.hbm.xml** 으로 파일이 생성된다. 클래스와 테이블간의 매핑을 시켜주는 역할이다.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" namespace="NHibernateDemo"
assembly="NHibernateDemo">
  <class name="NHibernateDemo.Entities.Shippers" table="Shippers">
    <id name="ShipperID" column="ShipperID" type="Int32" unsaved-value="-1" access="field.camelcase-
underscore">
      <generator class="identity" />
    </id>
    <property name="CompanyName" column="CompanyName" type="string" length="40" not-null="true"
access="field.camelcase-underscore"/>
    <property name="Phone" column="Phone" type="string" length="24" access="field.camelcase-
underscore"/>

  </class>
</hibernate-mapping>
```

<Generators 요소>

모든 generators 는 NHibernate.Id.IIdentifierGenerator 에서 구현된다.

- Increment : 다른 프로세스에서 데이터가 추가되지 않는 경우 유니크하게 사용되는 식별자로 클러스터에서는 사용을 금지한다.
- Identity : DB2, MySQL, MS SQL Server 그리고 Sybase 데이터 베이스에서 지원하는 식별자 컬럼이다.
- Sequence : DB2, PostgreSQL, Oracle 또는 Firebird 의 generator 에서 사용되는 순서다. 이 식별자는 Convert.ChangeType 속성 타입을 사용하여 변환한다.
- Hilo : hi/lo 알고리즘 사용으로 integral 타입의 식별자를 효과적으로 생성한다. hi/lo 알고리즘은 특정 데이터베이스에서 고유 식별자를 생성하기도 한다.
- Seqhilo : hi/lo 알고리즘을 사용하여 효과적인 integral 타입의 식별자를 생성하며 데이터베이스 순서가 명명된다.

●uuid.hex : System.Guid 와 ToString 메서드로 문자열 타입의 식별자를 생성한다. 문자열 길이는 구성 포맷에 따른다.

●uuid.string : 새로운 System.Guid 가 문자열로 변환되는 byte[]으로 생성한다.

●Guid : 새로운 System.Guid 식별자이다.

●guid.comb : Jimmy Nilsson 이 설명한(http://www.informit.com/articles/article.asp?p=25862) 새로운 System.Guid 을 생성하는 알고리즘을 사용한다.

●Native : 기본적인 데이터베이스의 기능에 따라 지정된다.

●Assigned : 응용프로그램에서 Save()이 호출되기 전에 객체 식별자를 지정한다.

●Foreign : 다른 연관된 객체의 식별자를 사용한다. 일반적으로 <one-to-one> 기본키가 사용된다.

<NHibernate Type>

Xml 파일 컬럼에 타입을 정의할 때 NHibernate Type 으로 정의해야 한다. 닷넷 타입과 데이터베이스 타입을 정리해놓은 것이다. 기본적으로 자주 사용되는 항목만 표시한다. 더 자세한 항목은 NHibernate 가이드 문서에 제공되고 있다.

NHibernate Type	.NET Type	Database Type
AnsiChar	System.Char	Db - Type.AnsiStringFixedL ength - 1 char
Boolean	System.Boolean	DbType.Boolean
Byte	System.Byte	DbType.Byte
Char	System.Char	Db- Type.StringFixedLengt h - 1 char
DateTime	System.DateTime	DbType.DateTime – ignores the milliseconds
Decimal	System.Decimal	DbType.Decimal
Double	System.Double	DbType.Double
Guid	System.Guid	DbType.Guid
Int16	System.Int16	DbType.Int16
Int32	System.Int32	DbType.Int32

Int64	System.Int64	DbType.Int64
PersistentEnum	A System.Enum	The DbType for the underlying value.
Single	System.Single	DbType.Single
Ticks	System.DateTime	DbType.Int64
TimeSpan	System.TimeSpan	DbType.Int64

NHibernate 경우 선수지식이 필요한 프레임워크이다. 이번 세션에서는 기본적으로 알아둬야 하는 사항을 몇 가지 정리했다.

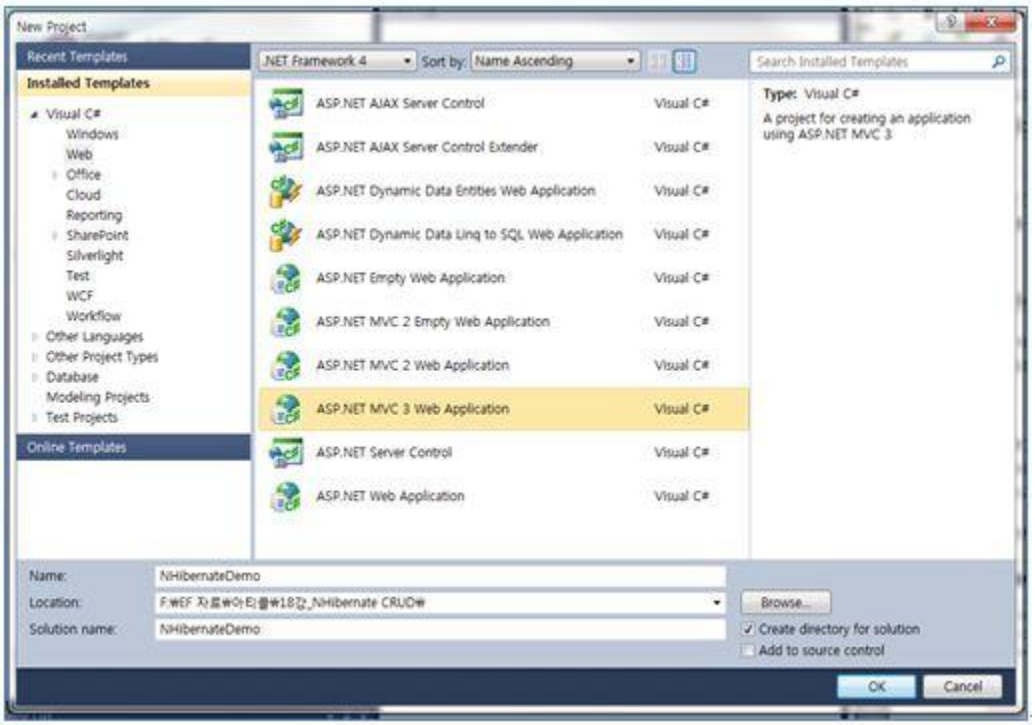
[NHibernate 강좌] 18. NHibernate 실전 – Object Relational Mapping

NHibernate 실전 – Object Relational Mapping

벌써 18 번째 세션을 진행하게 되었다. Entity Framework 4.0 에서 MyBatis.Net 그리고 NHibernate 까지 보통 ORM 프레임워크라고 불리는 것들에 대해서 살펴 보았다. 이번에 알아볼 내용은 ORM!! NHibernate 에서 말하는 Object Relational Mapping 에 대해서 알아보겠다.

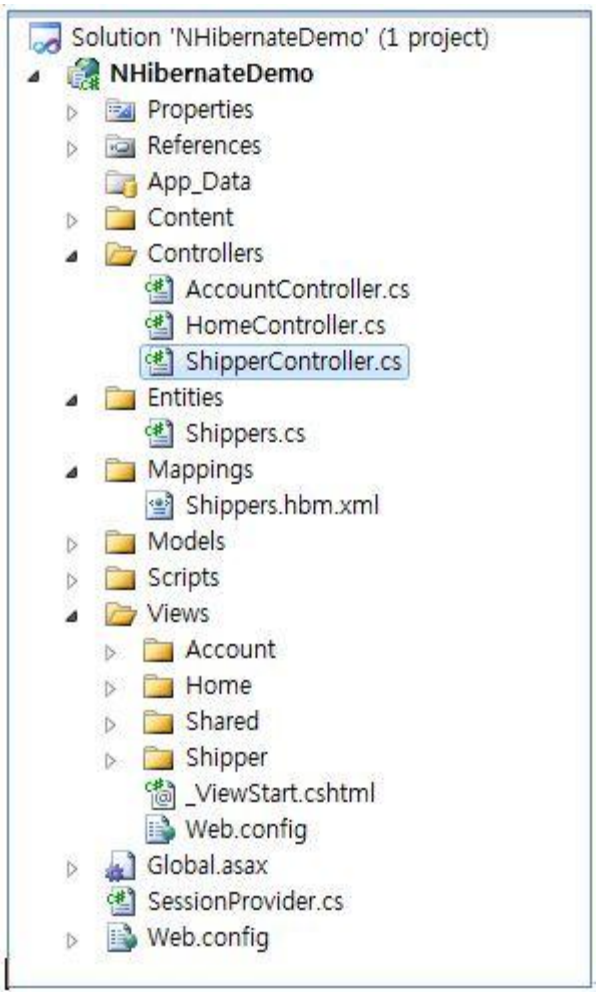
<실전! 프로젝트 기본 구성>

이번 세션부터 기본적인 사항들을 확인해서 실전으로 CRUD 까지 작업을 해보겠다. MVC Razor 웹 프로젝트를 하나 생성해준다. Entity Framework 를 되새기면서 해주시면 된다.



[그림 18-1] 웹 프로젝트 추가

MVC 웹 프로젝트를 다음과 같이 구성하려 한다. 미리 참고해준다.



[그림 18-2] 프로젝트 구성도

Web.config 에 Database Connection 정보를 입력해준다.

```
  <configSections>
    <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler,
NHibernate"/>
  </configSections>
  <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory name="NHibernateDemo">
      <property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
      <property name="connection.connection_string">
        Data Source=JISEON-PC₩JISEON2;Initial Catalog=Northwind;Persist Security Info=True;User
ID=sa;Password=P@ssw0rd;
      </property>
      <property name="adonet.batch_size">10</property>
      <property name="show_sql">true</property>
      <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
      <property name="use_outer_join">true</property>
      <property name="command_timeout">60</property>
      <property name="query.substitutions">true 1, false 0, yes 'Y', no 'N'</property>
      <property name="proxyfactory.factory_class">NHibernate.ByteCode.LinFu.ProxyFactoryFactory,
NHibernate.ByteCode.LinFu</property>
    </session-factory>
  </hibernate-configuration>
```

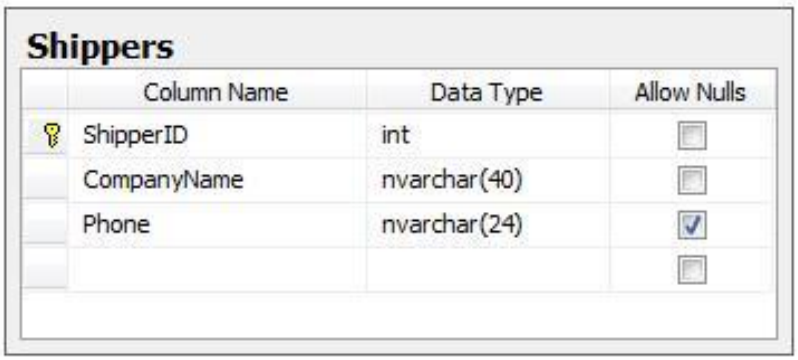
NHibernate 관련 DLL 을 참조해준다. NHibernate 에서 다운받은 폴더에 있다.

- NHibernate.dll
- Iesi.Collections.dll
- NHibernate.ByteCode.LinFu.dll
- LinFu.DynamicProxy.dll

<DataTable 과 Object>

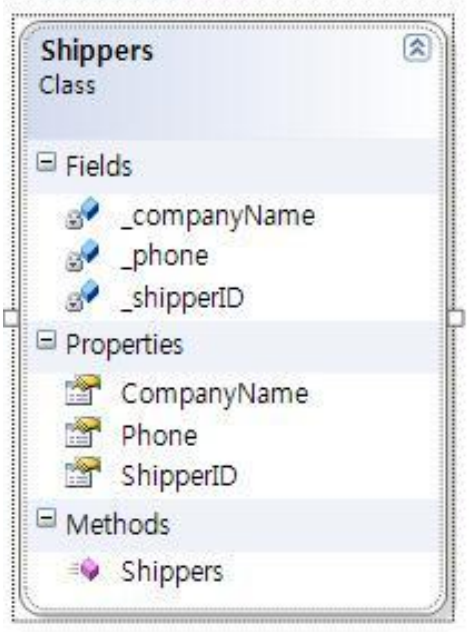
NHibernate 는 데이터 테이블을 기준으로 객체를 매핑한다. 매핑은 xml 파일에서 정의가 되어지며 앞 세션에서 말했듯이 <table>.hbn.xml 으로 생성된다.

Northwind 데이터베이스 Shippers 테이블 스키마이다.



[그림 18-3] Shippers 테이블 구조

테이블 구조에 맞게 Shippers 클래스를 구성해보겠다.



[그림 18-4] Shippers 테이블 구조

Shippers 클래스다.

```
public class Shippers
{
    private int _shipperID;
    private string _companyName;
    private string _phone;

    public Shippers()
    {
    }

    public virtual int ShipperID
    {
        get { return _shipperID; }
        set { _shipperID = value; }
    }

    public virtual string CompanyName
    {
        get { return _companyName; }
        set { _companyName = value; }
    }

    public virtual string Phone
    {
        get { return _phone; }
        set { _phone = value; }
    }
}
```

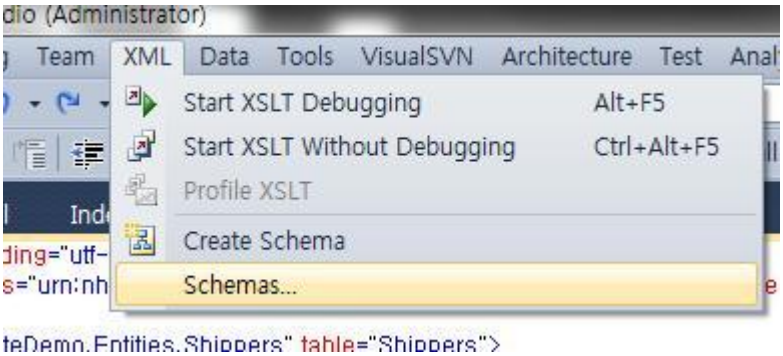
Database 의 테이블과 객체 매핑을 해주는 아래 정의된 xml 파일이다. 파일명은 테이블명을 앞에 두고 뒤에 hbn.xml 로 설정하면 된다. **"Shippers.hbm.xml"**

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" namespace="NHibernateDemo"
assembly="NHibernateDemo">

  <class name="NHibernateDemo.Entities.Shippers" table="Shippers">
    <id name="ShipperID" column="ShipperID" type="Int32" unsaved-value="-1" access="field.camelcase-
underscore">
      <generator class="identity" />
    </id>
    <property name="CompanyName" column="CompanyName" type="string" length="40" not-null="true"
access="field.camelcase-underscore"/>
    <property name="Phone" column="Phone" type="string" length="24" access="field.camelcase-
underscore"/>

  </class>
</hibernate-mapping>
```

Tip!

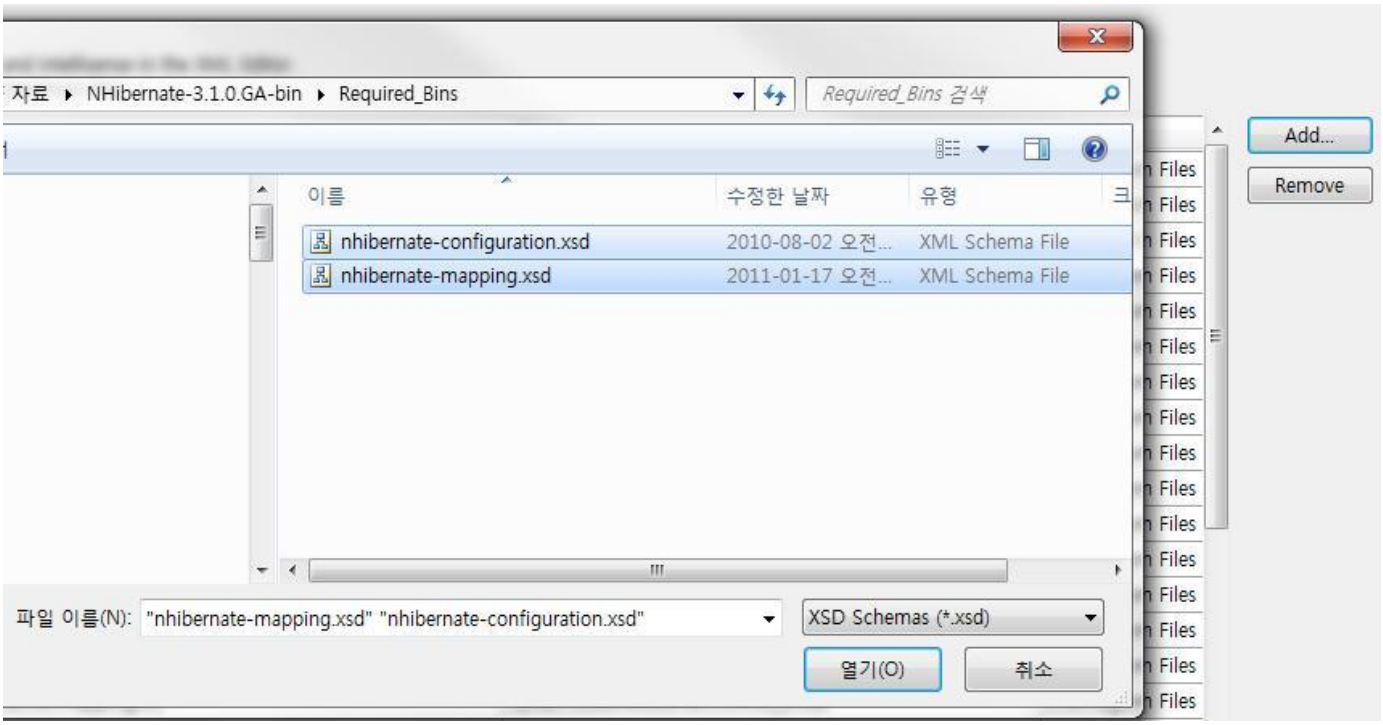
Visual Studio 에 익숙한 여러분은 실제로 위에 xml 파일들을 수동으로 설정하는 게 결코 쉬운 일은 아닐 겁니다. 저 역시도 xml 설정하는데 대소문자며 공백 등으로 어려움이 있었고요 NHibernate 에서 다운받은 파일 중에 Required_Bins 폴더에 xsd 스키마 파일이 존재한다.

Visual Studio 2010 에서 XML 메뉴를 클릭한다. Schemas.. 하위 메뉴를 클릭해서 위에서 봤던 스키마 파일을 추가하겠다.



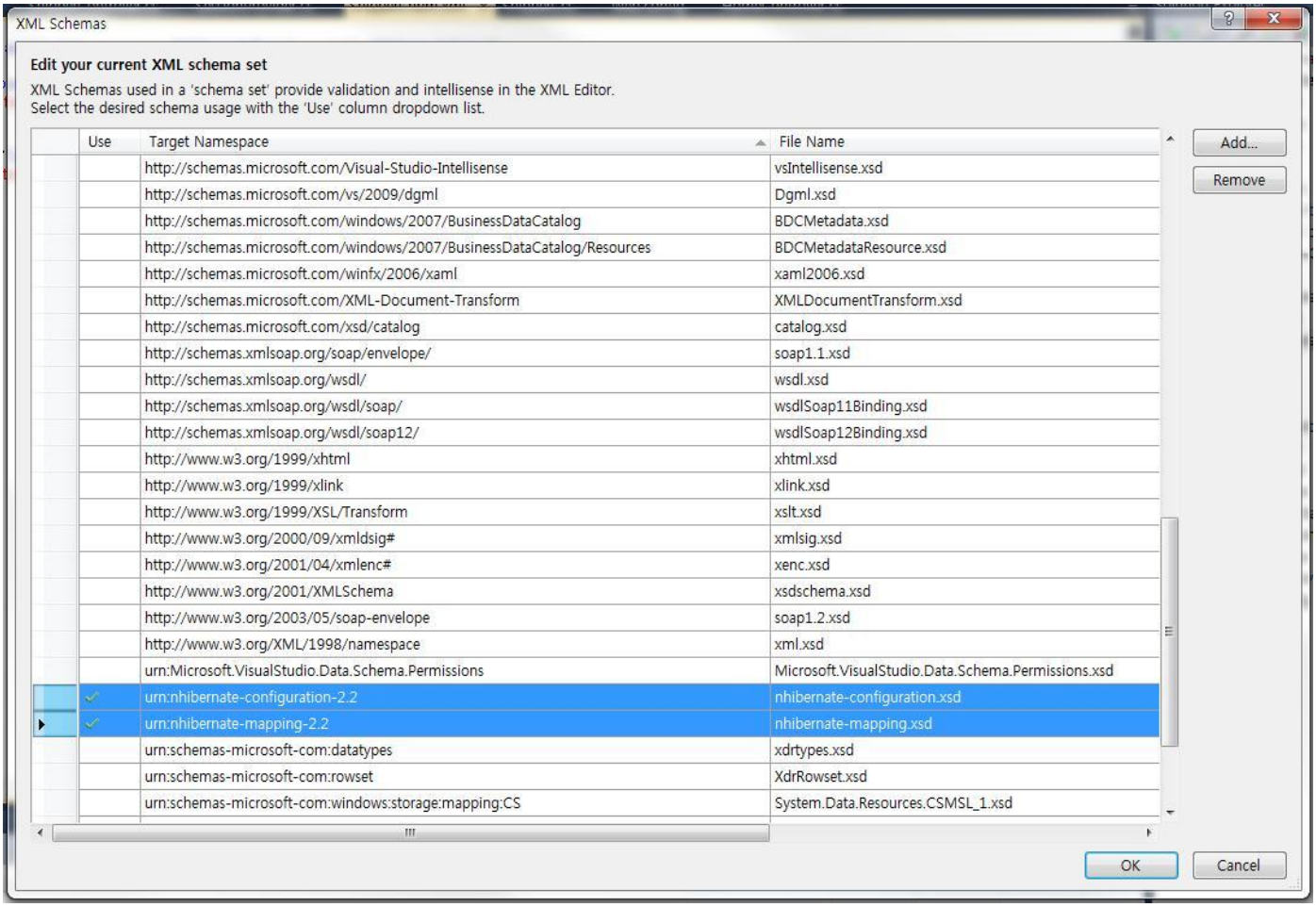
[그림 18-5] XML 메뉴

2 개 다 추가해준다.



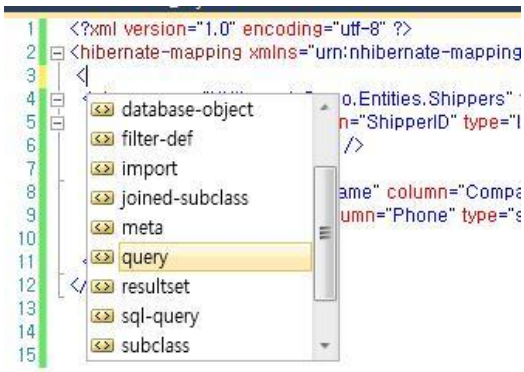
[그림 18-6] NHibernate 에서 제공하는 스키마

로드된 xsd 파일을 인텔리센스에 적용한다.



[그림 18-7] XML Schemas

이제 xml 파일에서 문자를 입력하면 아래와 같이 인텔리센스 기능이 바로 사용된다.



[그림 18-8] XML Intellisense

[NHibernate 강좌] 19. NHibernate 실전 – CRUD(1)

NHibernate 실전 - CRUD(1)

MyBatis.Net 경우 쿼리의 결과를 Object 로 가져온다면 NHibernate 는 데이터 테이블을 Object 로 가져오기 때문에 앞에서 살펴봤던 Entity Framework 처럼 이해하면 쉬울 듯 하다. 이번은 앞 세션에 이어서 Object 매핑 데이터를 조회하는 부분을 살펴 보겠다.

<NHibernate's ISession>

NHibernate 의 ISession 객체를 사용하여 데이터 저장소로 부터 원하는 데이터를 가져오겠다. ISessionFactory 로부터 ISession 을 얻을 수 있다.

다음 2 개의 네임스페이스를 추가해준다.

```
usingNHibernate;
usingNHibernate.Cfg;
```

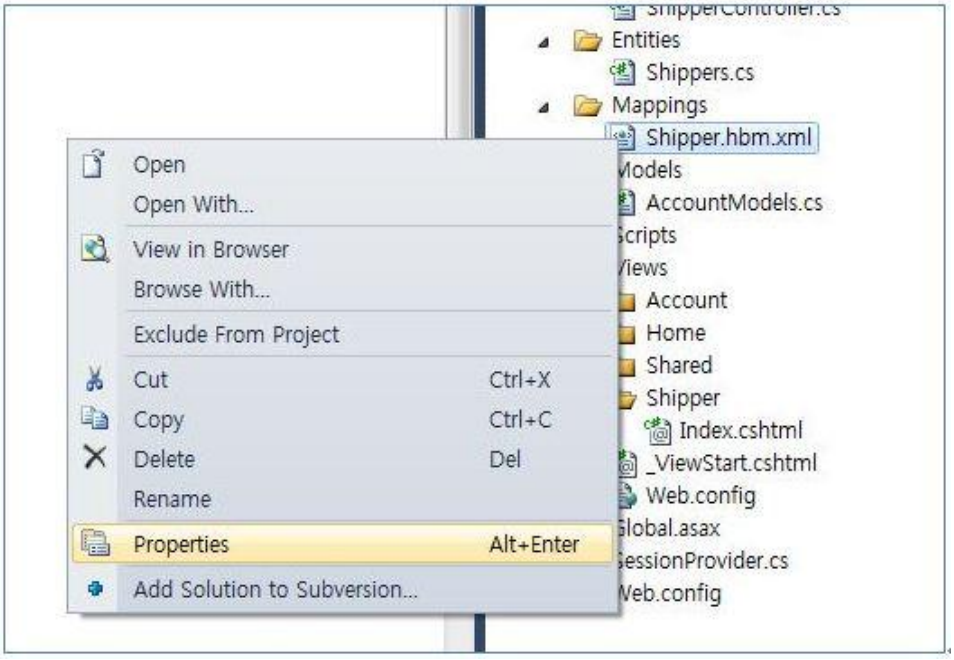
SessionProvider.cs 이다.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using NHibernate;
using NHibernate.Cfg;

namespace NHibernateDemo
{
   public class SessionProvider
   {
      private ISession _session;
      private ISessionFactory _sessionFac;
      private bool _reuseSession;

      #region Thread-safe Singleton

      public static SessionProvider Instance
      {
         get
         {
            return Nested.TheSingleton;
         }
      }

      private SessionProvider()
      {
         // Read the configuration
         Configuration cfg = new Configuration();
         cfg.AddAssembly("NHibernateDemo");

         _sessionFac = cfg.BuildSessionFactory();
         _reuseSession = false;
      }

      private class Nested
      {
         static Nested() { }
         internal static readonly SessionProvider
            TheSingleton = new SessionProvider();
      }

      #endregion
```

```
        /// <summary>
        /// Should the session be reused? If
        /// this property is false (default), a Session-Per-Request
        /// is applied.
        /// </summary>
        public bool ReuseSession
        {
            get
            {
                return _reuseSession;
            }
            set
            {
                _reuseSession = value;
            }
        }

        /// <summary>
        /// Get the current session. If the session is null,
        /// Open a new Session
        /// </summary>
        /// <returns>ISession-Object</returns>
        public ISession GetSession()
        {
            if (_session == null)
                _session = _sessionFac.OpenSession();

            return _session;

        }

        /// <summary>
        /// Close a Session, if the flag ReuseSession is false
        /// </summary>
        public void CloseSession()
        {
            if (_session != null)
            {
                if (!_reuseSession)
                {
                    _session.Close();
                    _session = null;
                }
            }
        }
    }
}
```

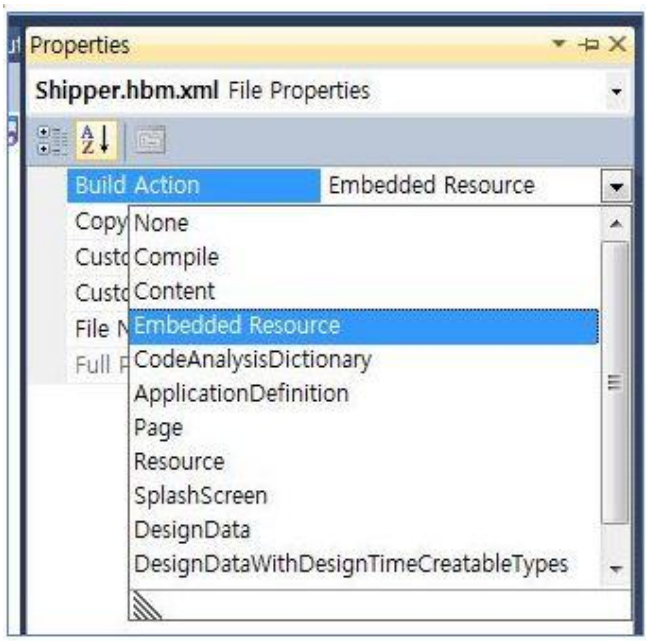
<빌드 리소스 포함>

매핑 Xml 파일은 빌드 시 리소스에 포함되어야 한다. 다음과 같이 설정해준다.



[그림 19-1] xml 속성 메뉴

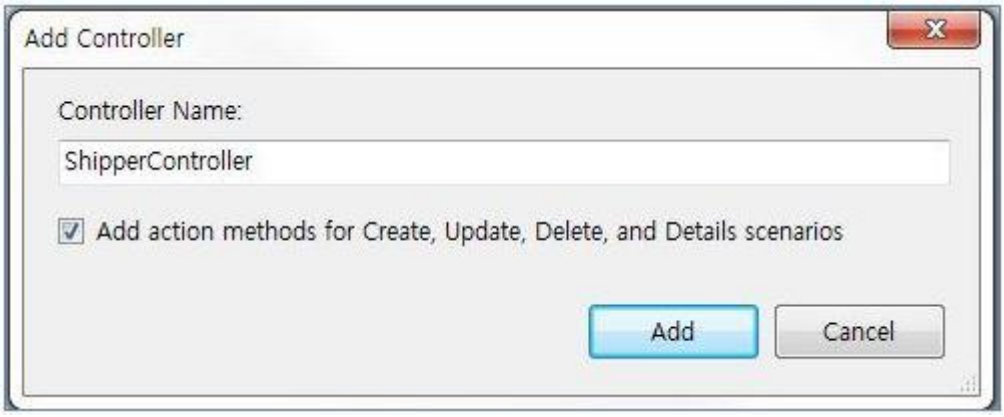
Xml 파일 속성에서 Build Action 에서 "리소스 포함"을 선택해준다. 그럼 빌드시 매핑 xml 파일이 자동으로 포함된다.



[그림 19-2] xml 속성

<조회 화면>

기다리고 기다리던 조회화면 이다. 이것을 하기 위해 우린 많은 것들을 설정했나 봅니다. MVC 프로젝트임으로 Controller 를 하나 추가하겠다. 기본적인 메서드들을 자동 추가하겠다.



[그림 19-3] Controller 추가

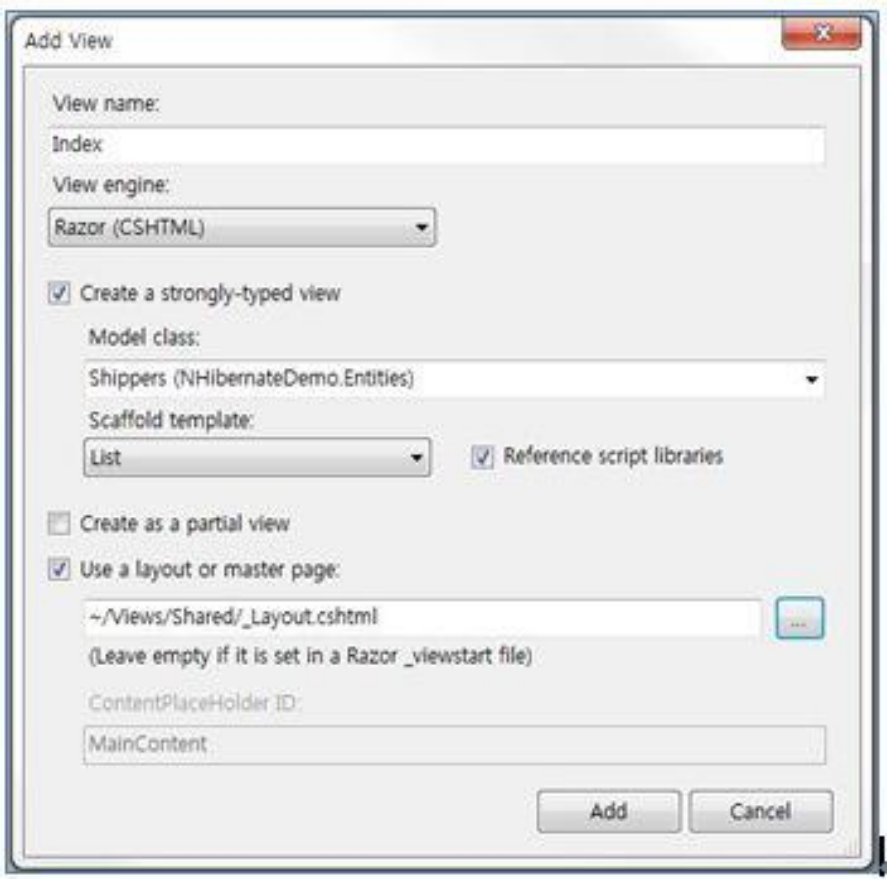
ShipperController 클래스에 다음 네임스페이스를 추가해준다.

```
using NHibernate;
using NHibernate.Cfg;
using NHibernateDemo.Entities;
```

리스트를 조회하는 코드를 다음과 같이 작성한다. 코드를 살펴보면 SessionProvider 에서 ISession 을 얻어와서 CreateCriteria 메서드를 통해서 기본 Shippers 객체를 반환한다. 받아온 객체를 바로 뷰에 반환해 준다.

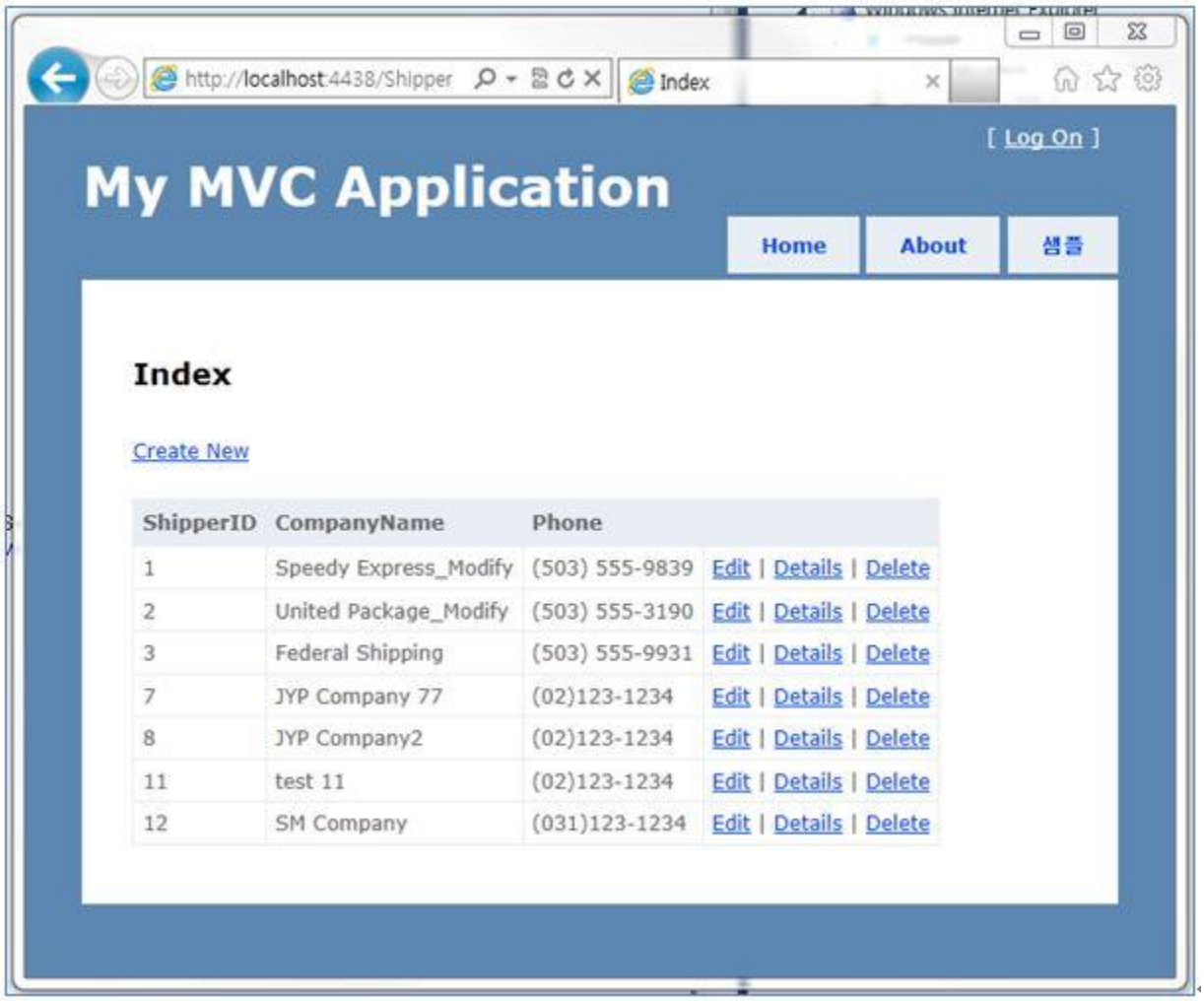
```
public ActionResult Index()
{
    IList<Shippers> list = null;
    try
    {
        ISession session = SessionProvider.Instance.GetSession();
        NHibernate.ICriteria ctiq = session.CreateCriteria(typeof(Shippers));
        list = ctiq.List<Shippers>();
    }
    catch (Exception ex)
    {
        throw;
    }

    return View(list);
}
```

Index 뷰를 생성해보겠다. 모델 클래스를 Shippers 로 설정 해주어야 한다.



[그림 19-4] View 추가

뷰에서 자동으로 리스트 코드가 작성된다. Shippers 클래스가 리스트에 매칭되었다. 웹 페이지를 실행해보겠다.



[그림 19-5] 리스트 화면 출력

[NHibernate 강좌] 20. NHibernate 실전 – CRUD(2)

NHibernate 실전 - CRUD(2)

NHibernate CRUD 두 번째 시간이다. 이번에는 비지니스 로직을 부분을 어떻게 구현하는지 살펴보겠다. CRUD 기본만 잡고가면 그 외 확장기능은 충분히 개발 가능할 것을 확신하기에 기본에 중점을 두고 진행한다.

<조회 화면>

조회하는 뷰페이지에 액션링크에 Shippers 의 Id 키값을 설정해 준다.

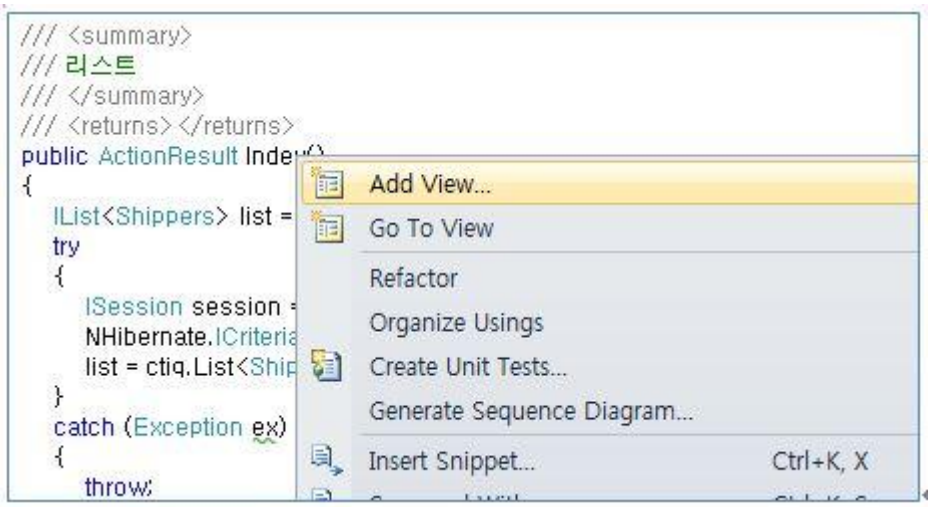
```
@Html.ActionLink("Edit", "Edit", new { id = item.ShipperID }) |
@Html.ActionLink("Details", "Details", new { id = item.ShipperID }) |
@Html.ActionLink("Delete", "Delete", new { id = item.ShipperID })
```

<상세 화면>

리스트에서 항목 클릭 시 이동되는 상세 화면을 구성해보겠다. 데이터의 키(id)값을 받아서 해당 데이터를 페이지에 출력하면 된다. Id 를 인자로 받아서 ISession Get 메서드를 사용하여 Shippers 의 특정데이터를 조회한다. 조회된 데이터는 뷰에 반환하여 페이지를 구성하게 된다.

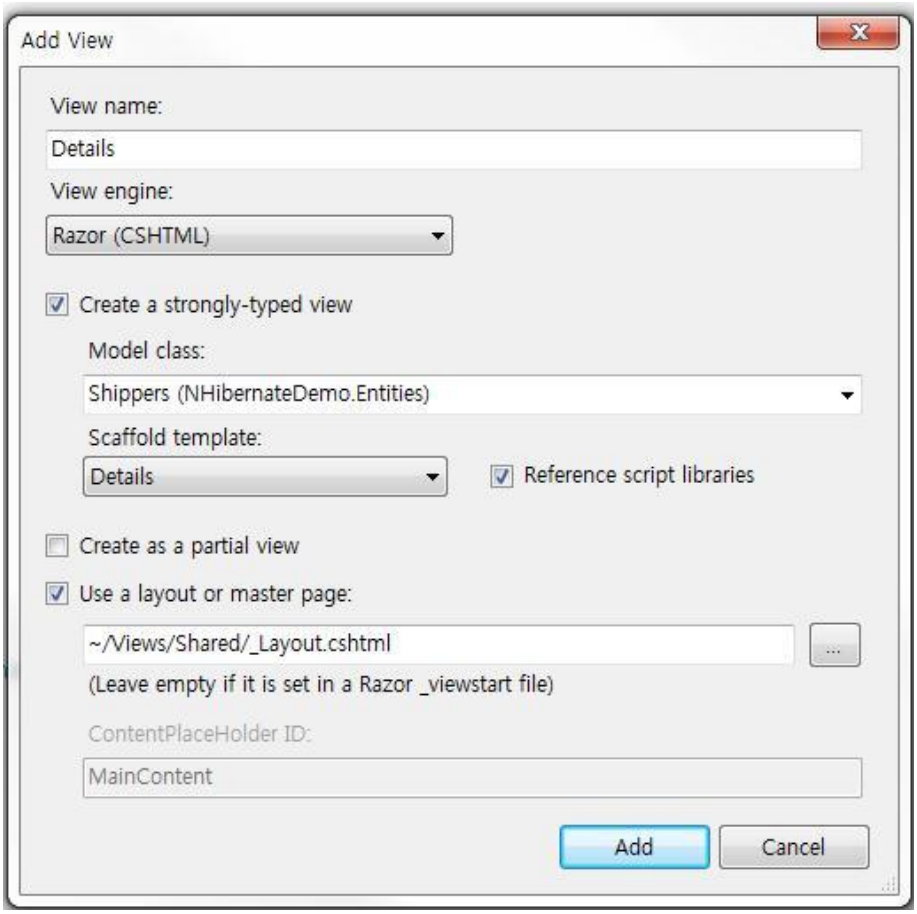
```
public ActionResult Details(int id)
{
    ISession session = SessionProvider.Instance.GetSession();
    Shippers shipper = session.Get<Shippers>(id);

    return View(shipper);
}
```

위의 Details 메서드가 구성되었다면 상세 뷰를 MVC 스캐폴딩 옵션을 사용하여 페이지를 생성 해보겠다. 메서드에서 마우스 오른쪽을 클릭하여 뷰 추가를 선택한다.



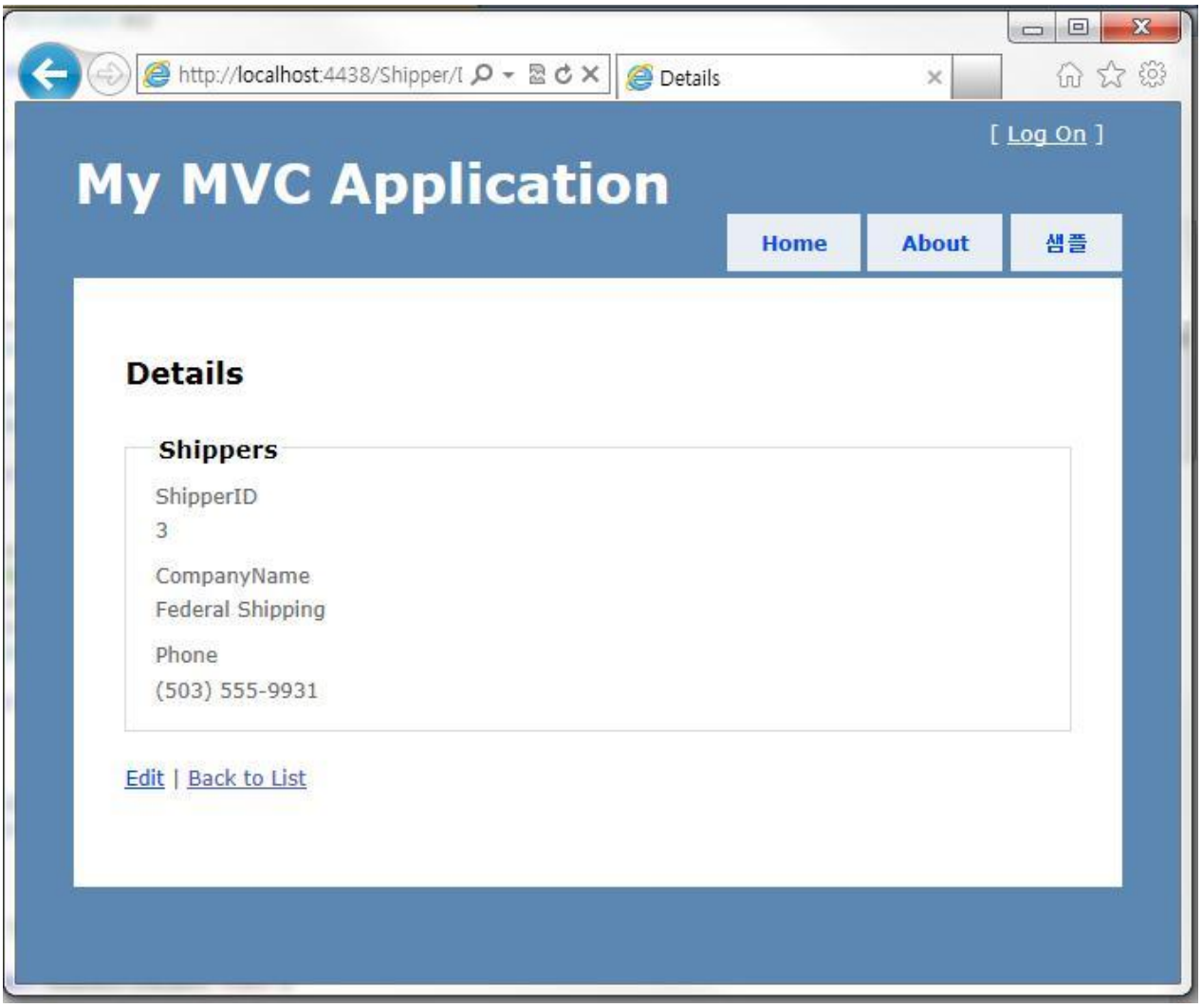
[그림 20-1] Controller 추가

화면에 매칭할 Model 을 선택하고 스캐폴링 옵션에 Details 로 선택한다.



[그림 20-2] View 추가

추가를 하면 Detail 이라는 뷰가 자동으로 생기며 뷰 코드를 보면 자동으로 데이터 상세 값이 출력되도록 구성된다. 실행 해보겠다.



[그림 20-3] 상세 화면

<신규생성>

뷰 구성은 위에 스캐폴딩을 이용해서 생성해주시면 된다. 그 부분은 생략하고 실제 로직을 살펴보겠다. 데이터 추가하는 메서드로 Shippers 객체를 받아서 바로 처리한다.

```
[HttpPost]
public ActionResult Create(Shippers shipper)
{
    ISession session = SessionProvider.Instance.GetSession();
    ITransaction trans = null;
    try
    {
        trans = session.BeginTransaction();
        session.Save(shipper);
        session.Flush();
        trans.Commit();
        return RedirectToAction("Index");
    }
    catch
    {
        trans.Rollback();
        return View();
    }
}
```

NHibernate 는 자체 Transaction 처리가 가능하다. ITransaction 객체를 사용하시면 된다. 데이터를 수정하거나 추가, 삭제처럼 데이터 변경이 발생하는 경우는 데이터베이스와 동기화를 위해 ISession.Flush()를 반드시 호출해줘야 한다.

<수정>

다음은 수정이다. 신규생성과 마찬가지로 Shippers 객체를 인자로 받아옵니다. ISession 의 Merge 메서드를 통해서 데이터 수정이 처리된다.

```
[HttpPost]
public ActionResult Edit(Shippers shipper)
{
    ISession session = SessionProvider.Instance.GetSession();
    ITransaction trans = null;
    try
    {
        trans = session.BeginTransaction();
        session.Merge(shipper);
        session.Flush();
        trans.Commit();
    }
    catch
    {
        trans.Rollback();
    }
    return RedirectToAction("Index");
}
```

<삭제>

마지막으로 데이터 삭제이다. Shippers 테이블은 ShipperID 를 Identity 로 가지고 있기때문에 id 를 인자로 받아와서 처리한다. 처리 후에 리스트화면으로 이동한다. 데이터 삭제 시에는 ISession 의 Delete 메서드를 이용한다.

```
public ActionResult Delete(int id)
{
    ISession session = SessionProvider.Instance.GetSession();
    Shippers shipper = session.Get<Shippers>(id);

    ITransaction trans = null;
    try
    {
        trans = session.BeginTransaction();
        session.Delete(shipper);
        session.Flush();
        trans.Commit();
        return RedirectToAction("Index");
    }
    catch
    {
        trans.Rollback();
        return View();
    }
}
```

소스코드 : NHibernateDemo.zip