**EF – Entity Framework**

**Конвенция – принимаемое соглашение.О книге**  
Пожалуйста, не стесняйтесь свободно делиться этой книгой со всеми. Последнюю версию оригинальной книги можно скачать с сайта: https://goalkicker.com/EntityFrameworkBook  
Книга *Entity Framework записки для Профессионалов* является собранной с Stack Overflow Documentation, содержание написано замечательными людьми Stack Overflow.

Текстовое содержимое выпущено под лицензией Creative Commons BY-SA, см. главу «Авторы» в конце книги, кем созданы различные главы. Распространяется право на использование изображений, если не указано иное под конкретным из них. Это неофициальная, свободная книга создана в целях обучения и не связанна как либо с официальной группой или компанией Entity Framework, Stack Overflow и иными организациями. Все торговые марки и знаки являются собственностью соответствующих владельцев компаний. Информация представленная в книге не гарантирует точность и корректность, вы используете ее на свой риск.

Пожалуйста, отправляйте свои отзывы и исправления на адрес: [web@petercv.com](mailto:web@petercv.com)

Перевел: alx.korolenok@gmail.com

**Глава 1: Начиная работу с Entity**  
**Framework**

**Версии:**  
1.0 2008-08-11  
4.0 2010-04-12  
4.1 2011-04-12  
4.1 Update 1 2011-07-25  
4.3.1 2012-02-29  
5.0 2012-08-11  
6.0 2013-10-17  
6.1 2014-03-17  
Core 1.0 2016-06-27

Замечания о выпусках: <https://msdn.microsoft.com/en-ca/data/jj574253.aspx>

**Часть 1.1: Что такое Entity Framework?**  
Написание и управление кода для доступа к данным с помощью ADO.Net является утомительной и монотонной работой.. Microsoft предоставляет фреймворк объектно-реляционного отображения (**O/RM framework)** называемый **"Entity Framework"** для автоматизации действий с базой данных для ваших приложений.

Entity framework это Объектно/Реляционное Отображение (Object/Relational Mapping (O/RM) framework).

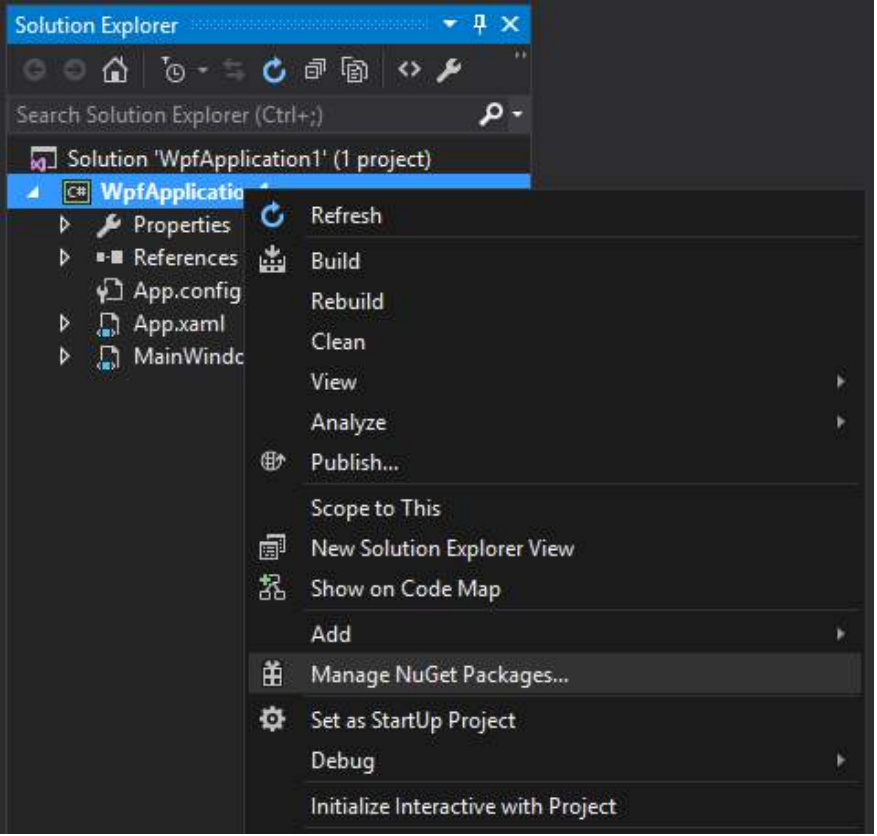
Это расширение ADO.NET, которое дает разработчикам автоматический механизм доступа и хранения данных в базах данных.

Что значит Объектно/Реляционное Отображение или **O/RM?**  
ORM это инструмент для хранения данных из объектов предметной области в реляционных базах данных, таких как MS SQL Server, в автоматическом режиме, без необходимости избыточного программирования.

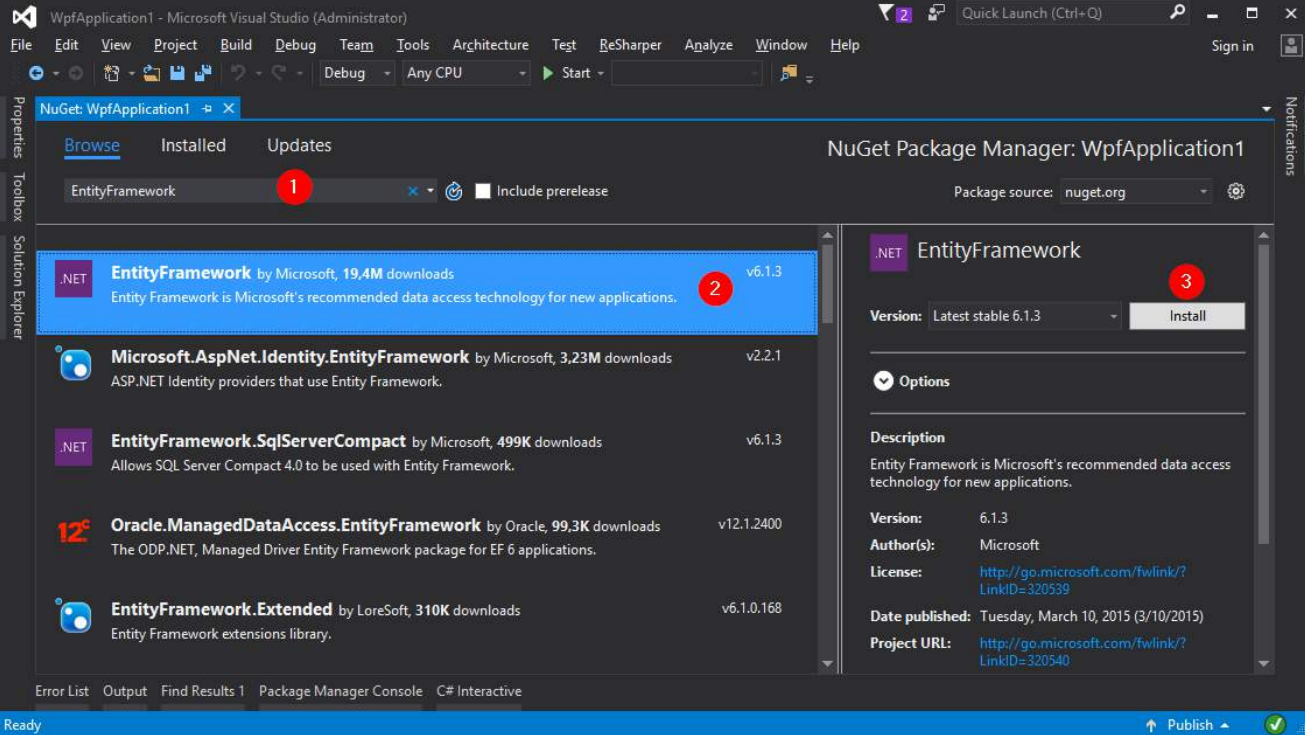
O/RM включает три главных части:  
1. Классы предметной области (сущности)   
2. Объекты реляционной базы данных  
3. Отображение информации о том как сущности отображаются на объекты базы данных (например таблицы, представления, хранимые процедуры).

ORM позволяет нам держать проект (дизайн) нашей базы данных отдельно от дизайна сущностей нашей предметной области. Это делает приложение более гибким к изменениям (ремонтопригодным) и расширяемым. Также это автоматизирует стандартные операции создания, чтения, обновления и удаления (CRUD), так что разработчику не нужно писать их самому.

**Часть 1.2: Установка Entity Framework через менеджер пакетов NuGet**  
В Visual Studio откройте окно обозревателя решений (**Solution Explorer)** после чего нажмите правой кнопкой мыши по своему проекту. Появится контекстное меню, где нужно выбрать *Manage NuGet Packages*:



В появившемся окне, в поле поиска нужно набрать EntityFramework.



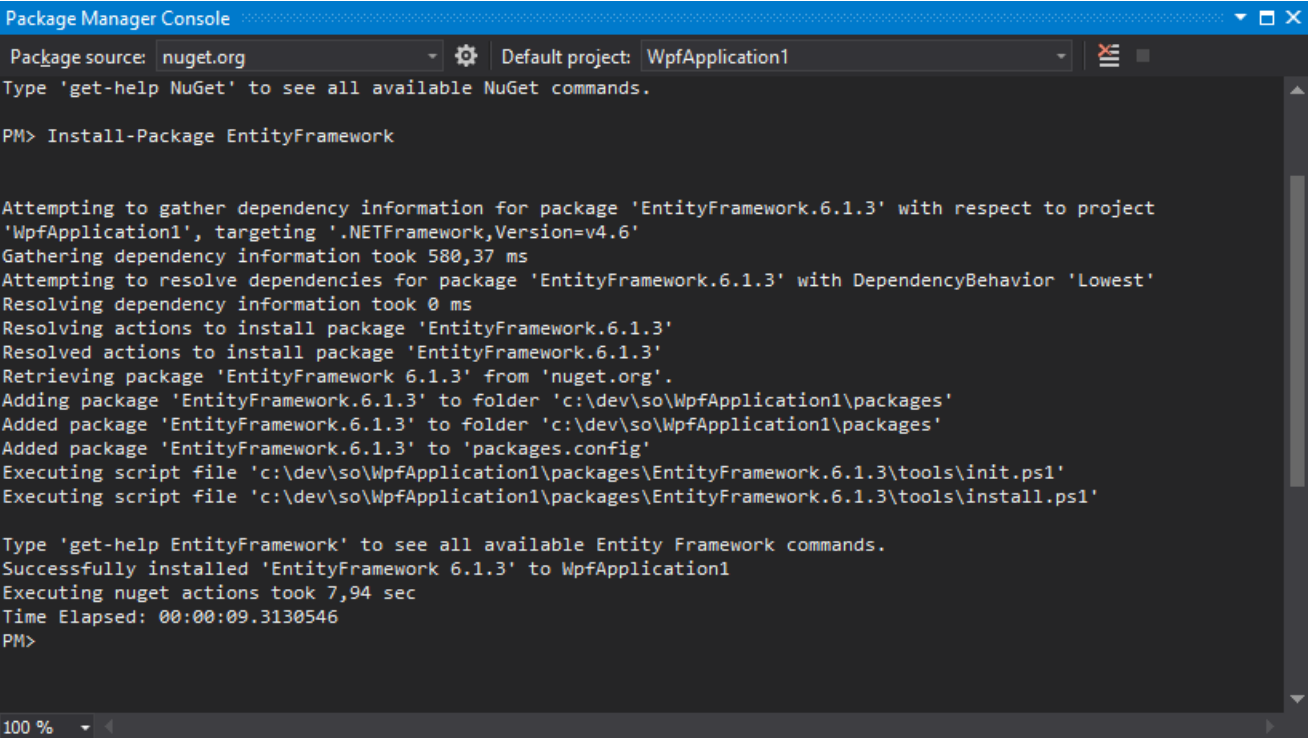
Далее нажмите кнопку установки (Install)

Также Entity Framework можно установить используя консоль менеджера пакетов (package manager console). Чтобы это сделать нужно открыть

*Tools menu -> NuGet Package Manager -> Package Manager Console*

и ввести команду:

Install-Package EntityFramework



Таким образом будет установлен EF и автоматически добавлена ссылка на сборку в вашем проекте.

**Часть 1.3: Использование Entity Framework из C# (Code First)**  
Code first позволяет вам создавать ваши сущности (классы) без использование графического дизайнера или файла отображения данных .edmx. Этот подход называется «Сначала код» потому, что вы можете создать свои модели первыми и *Entity framework* создаст вам базу данных в соответствии с отображением автоматически. Или вы можете также использовать этот подход с существующей базой данных, который называется «Сначала код, с существующей БД» (*code first with existing database).*

Для примера, если вы хотите держать в таблице список планет:

public class Planet  
{  
 public string Name { get; set; } // Название  
 public decimal AverageDistanceFromSun { get; set; } // Среднее расстояние от солнца  
}

Теперь создайте ваш контекст данных, который является мостом между вашими классами сущностей и базой данных. Определите в этом контексте один или несколько наборов свойств, закрытый типом необходимой сущности. В данном случае **Planet**.

DbSet<> свойства:  
using System.Data.Entity;  
public class PlanetContext : DbContext  
{  
 public DbSet<Planet> Planets { get; set; }  
}

Мы может использовать этот контекст сделав следующее:

using(var context = new PlanetContext())  
{  
 var jupiter = new Planet  
 {  
 Name = "Jupiter",  
 verageDistanceFromSun = 778.5  
 };  
 context.Planets.Add(jupiter);  
 context.SaveChanges();  
}

В данном примере мы создаем новую планету со свойством Name, значение которого "Jupiter" и свойством AverageDistanceFromSun со значением 778.5

После чего мы можем добавить эту планету в наш контекст используя метод набора свойств (DbSet) Add() . Чтобы зафиксировать наши изменения в базе данных используем метод SaveChanges().

Или мы можем извлечь строки из базы данных:

using(var context = new PlanetContext())  
{  
 var jupiter = context.Planets.Single(p => p.Name == "Jupiter");  
 Console.WriteLine($"Jupiter is {jupiter.AverageDistanceFromSun} million km from the sun.");  
}

**Глава 2: конвенции Code First**  
**Часть 2.1: конвенции удаления**

Вы можете удалить любую из конвенций определенных в пространстве имен System.Data.Entity.ModelConfiguration.Conventions  
путем переопределения метода OnModelCreating.

В следующем примере будет удалено соглашение об именовании множественных сущностей (PluralizingTableNameConvention).  
  
public class EshopContext : DbContext  
{  
 public DbSet<Product> Products { set; get; }  
 . . .  
 protected override void OnModelCreating(DbModelBuilder modelBuilder)  
 {  
 modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();  
 }  
}

По умолчанию EF будет создавать таблицы базы данных с именем класса после которого ставится суффикс 's' (обозначая множественное число). В данном примере, Code First сконфигурирован таким образом, что будет игнорировать данную конвенцию (PluralizingTableName), и вместо таблицы с именем **dbo.Products** будет создана **dbo.Product**.

**Часть 2.2: конвенция первичного ключа**  
По умолчанию свойство, являющееся первичным ключом называется “ID” (не чувствительно к регистру), или по названию класса, за которым следует "ID". Если тип данных первичного ключа является числовым или GUID, то поле базы данных будет сконфигурировано как идентифицирующее (первичный ключ).

Простой пример:

public class Room  
{  
 *// Primary key*  
 public int RoomId{ get; set; }  
 ...  
}

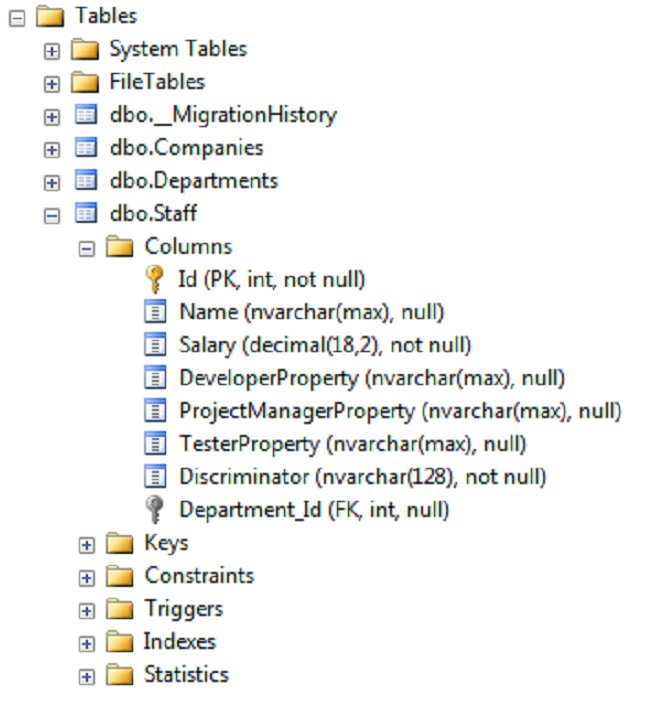
**Часть 2.3: Раскрытие типа**  
По умолчанию Code First включает в модель следующее:  
1. Типы, определенные как наборы свойств (DbSet) в классе контекста.  
2. Ссылочные типы включенные в типы сущностей, даже если они определены в разных сборках.   
3. Унаследованные классы, даже если базовый класс определен как свойство DbSet.

В примере ниже мы добавим компанию как набор свойств в наш класс контекста:

public class Company  
{  
 public int Id { set; get; }  
 public string Name { set; get; }  
 public virtual ICollection<Department> Departments { set; get; }  
}  
public class Department  
{  
 public int Id { set; get; }  
 public string Name { set; get; }  
 public virtual ICollection<Person> Staff { set; get; }  
}

[Table("Staff")]  
public class Person  
{  
 public int Id { set; get; }  
 public string Name { set; get; }  
 public decimal Salary { set; get; }  
}  
public class ProjectManager : Person  
{  
 public string ProjectManagerProperty { set; get; }  
}  
public class Developer : Person  
{  
 public string DeveloperProperty { set; get; }  
}  
public class Tester : Person  
{  
 public string TesterProperty { set; get; }  
}  
public class ApplicationDbContext : DbContext  
{  
 public DbSet<Company> Companies { set; get; }  
}

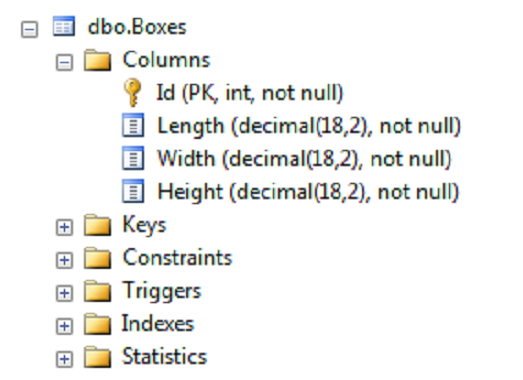
Как мы можем видеть, все наши классы включены в нашу модель:



**Часть 2.4: конвенция десятичных свойств**

По умолчанию EF отображает свойства типа decimal в поля таблиц базы данных типа decimal(18,2).

public class Box  
{  
 public int Id { set; get; }  
 public decimal Length { set; get; }  
 public decimal Width { set; get; }  
 public decimal Height { set; get; }  
}

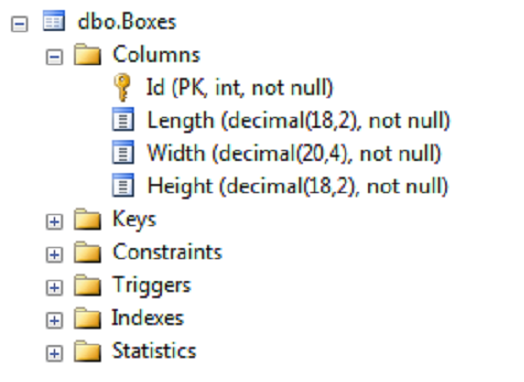


Мы можем менять точность таких (десятичных) свойств:

1. Используя Fluent API:

protected override void OnModelCreating(DbModelBuilder modelBuilder)  
{  
 modelBuilder.Entity<Box>().Property(b => b.Width).HasPrecision(20, 4);  
}

Только свойство "Width" будет отображено в decimal(20, 4).



1. Изменить конвенцию:

protected override void OnModelCreating(DbModelBuilder modelBuilder)  
{  
 modelBuilder.Conventions.Remove<DecimalPropertyConvention>();  
 modelBuilder.Conventions.Add(new DecimalPropertyConvention(10, 4));  
}

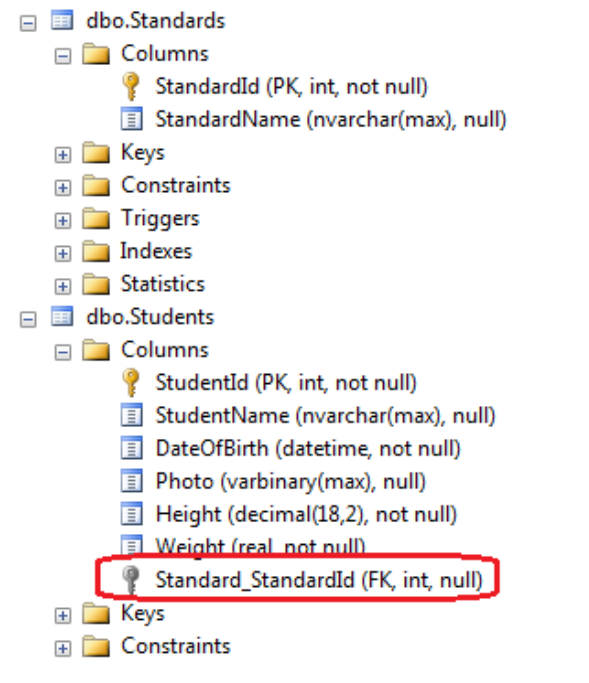
Каждое свойство типа decimal будет отображено в тип столбца decimal(10,4).

**Часть 2.5: конвенция отношений**   
Code First делает вывод об отношениях между двумя сущностями используя навигационное свойство.

Это навигационное свойство может быть простым ссылочным типом или типом коллекции. К примеру, мы можем определить навигационное свойство **Standard** в классе Student и навигационное свойство типа ICollection в классе Standard. Так, Code First автоматически создаст отношение один-ко-многим между табицами Standards и Students. В БД, в таблице Students будет создан внешний ключ Standard\_StandardId.

public class Student  
{  
 public int StudentID { get; set; }  
 public string StudentName { get; set; }  
 public DateTime DateOfBirth { get; set; }  
 *//Navigation property*  
 public Standard Standard { get; set; }  
}  
public class Standard  
{  
 public int StandardId { get; set; }  
 public string StandardName { get; set; }  
 *//Collection navigation property*  
 public IList<Student> Students { get; set; }  
}

В вышеуказанных сущностях создано отношение использующее внешний ключ Standard\_StandardId.



**Часть 2.6: конвенция внешнего ключа**   
Если класс A состоит в отношении с классом B и класс B имеет свойство с таким же именем и типом как первичный ключ класса A, тогда EF автоматически предполагает, что свойство является внешним ключом.

public class Department  
{  
 public int DepartmentId { set; get; }  
 public string Name { set; get; }  
 public virtual ICollection<Person> Staff { set; get; }  
}  
public class Person  
{  
 public int Id { set; get; }  
 public string Name { set; get; }  
 public decimal Salary { set; get; }  
 public int DepartmentId { set; get; }  
 public virtual Department Department { set; get; }  
}  
В данном случае DepartmentId является внешним ключом без явной спецификации.

Глава 3: Code First Аннотация данных

Аннотация данных (DataAnnotation) – способ конфигурирования отображения через аннотирующие атрибуты.

Часть 3.1: Атрибут [Column]public class Person  
{  
 public int PersonID { get; set; }  
 [Column("NameOfPerson")]  
 public string PersonName { get; set; }  
}

Сообщает EF использовать конкретное имя столбца вместо используемого имени свойства. Вы также можете указать тип данных для столбца и номер по порядку столбца в таблице.

Tells Entity Framework to use a specific column name instead using the name of the property. You can also specify  
the database data type and the order of the column in table:  
[Column("NameOfPerson", TypeName = "varchar", Order = 1)]  
public string PersonName { get; set; }

Часть 3.2: Атрибут [DatabaseGenerated]Уточняет как база данных генерирует значения для свойства. Возможны три варианта:

1. None указывает что значения не могут быть сгенерированы базой данных.  
2. Identity уточняет столбец является идентифицирующим, который типично column is an identity column, which is typically used for integer primary keys.  
3. Computed specifies that the database generates the value for the column.  
If the value is anything other than None, Entity Framework will not commit changes made to the property back to  
the database.  
By default (based on the StoreGeneratedIdentityKeyConvention) an integer key property will be treated as an  
identity column. To override this convention and force it to be treated as a non-identity column you can use the  
DatabaseGenerated attribute with a value of None.  
using System.ComponentModel.DataAnnotations.Schema;  
public class Foo  
{  
[Key]  
public int Id { get; set; } // identity (auto-increment) column}  
public class Bar  
{  
[Key]  
[DatabaseGenerated(DatabaseGeneratedOption.None)]  
public int Id { get; set; } // non-identity column}  
The following SQL creates a table with a computed column:  
CREATE TABLE [Person] (  
Name varchar(100) PRIMARY KEY,  
DateOfBirth Date NOT NULL,  
Age AS DATEDIFF(year, DateOfBirth, GETDATE())  
)  
GoalKicker.com – Entity Framework Notes for Professionals 12  
GO  
To create an entity for representing the records in the above table, you would need to use the DatabaseGenerated  
attribute with a value of Computed.  
[Table("Person")]  
public class Person  
{  
[Key, StringLength(100)]  
public string Name { get; set; }  
public DateTime DateOfBirth { get; set; }  
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]  
public int Age { get; set; }  
}  
Section 3.3: [Required] attributeWhen applied to a property of a domain class, the database will create a NOT NULL column.  
using System.ComponentModel.DataAnnotations;  
public class Person  
{  
public int PersonID { get; set; }  
[Required]  
public string PersonName { get; set; }  
}