Аннотации

Последнее обновление: 26.09.2016

Аннотации представляют настройку сопоставления моделей и таблиц с помощью атрибутов. Большинство классов аннотаций располагаются в пространстве `System.ComponentModel.DataAnnotations`, которое нам надо подключить в файл c# перед использованием аннотаций.

Настройка ключа

Для установки свойства в качестве первичного ключа применяется атрибут `[Key]`:

```
public class Phone
{
    [Key]
    public int Ident { get; set; }
    public string Name { get; set; }
}
```

Теперь свойство Ident будет рассматриваться в качестве первичного ключа.

Чтобы установить ключа в качестве идентификатора, можно использовать атрибут `DatabaseGenerated(DatabaseGeneratedOption.Identity)`:

```
[Key,DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Ident { get; set; }
```

Если свойство, помеченное атрибутом `Key`, представляет тип `int`, то к нему атрибут `DatabaseGenerated(DatabaseGeneratedOption.Identity)` будет применяться автоматически, и его можно в принципе не указывать.

Атрибут Required

Атрибут **Required** указывает, что данное свойство обязательно для установки, то есть будет иметь определение NOT NULL в БД:

```
public class Phone
{
    [Key]
    public int Ident { get; set; }
        [Required]
    public string Name { get; set; }
}
```

Если мы не установим свойство Name у объекта Phone и попытаемся добавить этот объект в бд, то получим ошибку. А столбец Name будет определен как NOT NULL.

MaxLength и MinLength

Атрибуты **MaxLength** и **MinLength** устанавливают максимальное и минимальное количество символов в строке-свойстве:

```
public class Phone
{
    [Key]
    public int Ident { get; set; }
    [MaxLength(20)]
    public string Name { get; set; }
}
```

Атрибут NotMapped

По умолчанию все публичные свойства сопоставляются с определенными столбцами в таблицами. Но такое поведение не всегда необходимо. Иногда требуется, наоборот, исключить определенное свойство, чтобы для него не создавался столбец в таблице. И для этих целей есть атрибут NotMapped:

```
public class Phone
{
    [Key]
    public int Ident { get; set; }
        [Required]
    public string Name { get; set; }
        [NotMapped]
    public int Discount { get; set; }
}
```

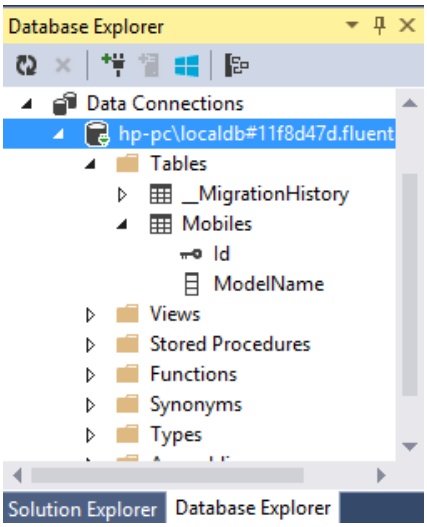
Чтобы задействовать атрибут NotMapped, надо подключить пространство имен `System.ComponentModel.DataAnnotations.Schema`

Сопоставление с таблицей и столбцами

Entity Framework при создании и сопоставлении таблиц и столбцов использует имена моделей и их свойств. Но мы можем переопределить это поведение с помощью атрибутов **Table** и **Column**:

```
[Table("Mobiles")]
public class Phone
{
    public int Id { get; set; }
    [Column("ModelName")]
    public string Name { get; set; }
}
```

Теперь сущность Phone будет сопоставляться с таблицей Mobiles, а свойство Name со столбцом ModelName:



Установка внешнего ключа

Чтобы установить внешний ключ для связи с другой сущностью, используется атрибут **ForeignKey**:

```
public class Phone
{
```

```
    public int Id { get; set; }
    public string Name { get; set; }

    public int? CompId { get; set; }
    [ForeignKey("CompId")]
    public Company Company { get; set; }
}

public class Company
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

В данном случае внешним ключом для связи с моделью Company будет служить свойство CompId.

Настройка индекса

Для установки индекса для столбца к соответствующему свойству модели применяется атрибут **[Index]**

```
public class Phone
{
        [Index]
    public int Id { get; set; }
    public string Name { get; set; }
}
```

ConcurrencyCheck

Атрибут **ConcurrencyCheck** позволяет решить проблему параллелизма, когда с одной и той же записью в таблице могут работать одновременно несколько пользователей. Например:

```
public class Phone
{
    public int Id { get; set; }
        [ConcurrencyCheck]
    public string Name { get; set; }
}
```

Теоретически возможна ситуация, когда два пользователя пытаются изменить значение, например:

```
using(FluentContext db = new FluentContext())
{
    Phone phone = db.Phones.Find(1);
    phone.Name = "Nokia N9";
    db.SaveChanges();
}
```

В обычном режиме Entity Framework при обновлении смотрит на Id и если Id записи в таблице совпадает с Id в передаваемой модели phone, то строка в таблице обновляется. При использовании атрибута ConcurrencyCheck EF смотрит не только на Id, но и на исходное значение свойства Name. И если оно совпадает с тем, что имеется в таблице, то запись обновляется. Если же не совпадает (то есть кто-то уже успел обновить), то EF генерирует исключение **DbUpdateConcurrencyException**.

Назад **Содержание** **Вперед**