Навигационные свойства и загрузка данных

Последнее обновление: 28.09.2016

В предыдущей теме в качестве модели был использован класс Player, представляющий футболиста и содержащий четыре свойства. Однако эта была очень простая модель. В реальности в нашей базе данных может быть не одна, а несколько таблиц, которые связаны между собой различными связями.

Допустим, для каждого футболиста может быть определена футбольная команда, в которой он играет. И, наоборот, в одной футбольной команде могут играть несколько футболистов. То есть в данном случае у нас связь **один-ко-многим (one-to-many)**.

Например, у нас определен следующий класс футбольной команды Team:

```
class Team
{
    public int Id { get; set; }
    public string Name { get; set; } // название команды
    public string Coach { get; set; } // тренер

        public ICollection<Player> Players { get; set; }
}
```

А класс Player, описывающий футболиста, мог бы выглядеть следующим образом:

```
class Player
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Position { get; set; }
    public int Age { get; set; }

    public int? TeamId { get; set; }
    public Team Team { get; set; }
}
```

Кроме обычных свойств типа Name, Position и Age здесь также определен внешний ключ. Внешний ключ состоит из обычного свойства и навигационного.

Свойство `public Team Team { get; set; }` в классе Player называется **навигационным свойством** - при получении данных об игроке оно будет автоматически получать данные из БД.

Аналогично в классе Team также имеется навигационное свойство - `Players`, через которое мы можем получать игроков данной команды.

Вторая часть внешнего ключа - свойство TeamId. Чтобы в связке с навигационным свойством образовать внешний ключ оно должно принимать одно из следующих вариантов имени:

- *Имя_навигационного_свойства+Имя ключа из связанной таблицы* - в нашем случае имя навигационного свойства Team, а ключа из модели Team - Id, поэтому в нашем случае нам надо обозвать свойство TeamId, что собственно и

было сделано в вышеприведенном коде.

- *Имя_класса_связанной_таблицы+Имя ключа из связанной таблицы* - в нашем случае класс Team, а ключа из модели Team - Id, поэтому опять же в этом случае получается TeamId.

Как уже было сказано, внешний ключ позволяет получать связанные данные. Например, после генерации базы данных с помощью Code First таблица Players будет иметь следующее определение:

```
CREATE TABLE [dbo].[Players] (
    [Id]       INT            IDENTITY (1, 1) NOT NULL,
    [Name]     NVARCHAR (MAX) NULL,
    [Position] NVARCHAR (MAX) NULL,
    [Age]      INT            NOT NULL,
    [TeamId]   INT            NULL,
    CONSTRAINT [PK_dbo.Players] PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_dbo.Players_dbo.Teams_TeamId] FOREIGN KEY ([TeamId]) REFERENCES [dbo].[Teams] ([Id])
);
```

При определении внешнего ключа нужно иметь в виду следующее. Если тип обычного свойства во внешнем ключе определяется как `int?`, то есть допускает значения `null`, то при создании базы данных соответствующее поле так будет принимать значения NULL: `[TeamId] INT NULL`.

Однако если мы изменим в классе Player тип TeamId на просто `int`: `public int TeamId { get; set; }`, то в этом случае соответствующее поле имело бы ограничение `NOT NULL`, а внешний ключ определял бы каскадное удаление:

```
CREATE TABLE [dbo].[Players] (
    [Id]       INT            IDENTITY (1, 1) NOT NULL,
    [Name]     NVARCHAR (MAX) NULL,
    [Position] NVARCHAR (MAX) NULL,
    [Age]      INT            NOT NULL,
    [TeamId]   INT            NOT NULL,
    CONSTRAINT [PK_dbo.Players] PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_dbo.Players_dbo.Teams_TeamId] FOREIGN KEY ([TeamId]) REFERENCES [dbo].[Teams] ([Id]) ON DELETE
CASCADE
);
```

Способы загрузки и получения связанных данных

В Entity Framework есть три способа загрузки данных:

- **eager loading**("жадная загрузка")
- **explicit loading**("явная загрузка")
- **lazy loading**("ленивая загрузка")

Eager Loading

Суть Eager Loading заключается в том, чтобы использовать для подгрузки связанных по внешнему ключу данных метод **Include**. Например, получим всех игроков с их командами:

```
using(SoccerContext db = new SoccerContext())
{
        var players = db.Players.Include(p=>p.Team).ToList();
        foreach(Player p in players)
        {
                MessageBox.Show(p.Team.Name);
        }
}
```

Без использования метода Include мы бы не могли бы получить связанную команду и ее свойства: `p.Team.Name`

Соответственно чтобы подгрузить к командам все данные по игрокам, мы можем написать так:

```
using(SoccerContext db = new SoccerContext())
{
        var teams = db.Teams.Include(t=>t.Players).ToList();
```

```
    foreach (var t in teams)
    {
        Console.WriteLine($"{t.Name}");
        foreach(var p in t.Players)
            Console.WriteLine($"{p.Name}");
    }
}
```

Explicit Loading

Явная загрузка предусматривает применение метода **Load()** для загрузки данных в контекст. Например:

```
using(SoccerContext db = new SoccerContext())
{
        var p = db.Players.FirstOrDefault();
    db.Entry(p).Reference("Team").Load();
    Console.WriteLine($"{p.Name} - {p.Team.Name}");


    var t = db.Teams.FirstOrDefault();
    db.Entry(t).Collection("Players").Load();
    Console.WriteLine($"{t.Name}");
    foreach(var pl in t.Players)
        Console.WriteLine($"{pl.Name}");
}
```

Чтобы подгрузить данные, здесь идет обращение к методу `db.Entry()`, в который передается нужный объект. Для подгрузки связанного объекта, который не представляет коллекцию, используется метод **Reference()**. В этот метод переается навигационное свойство, по которому надо подгрузить данные.

Если связанные объект представляет коллекцию, то применяется метод `Collection()`, в который также передается навигационное свойство в виде строки.

Lazy Loading

Еще один способ представляет так называемая "ленивая загрузка" или **lazy loading**. При таком способе подгрузки при первом обращении к объекту, если связанные данные не нужны, то они не подгружаются. Однако при первом же обращении к навигационному свойству эти данные автоматически подгружаются из бд.

При использовании ленивой загрузки надо иметь в виду некоторые моменты при объявлении классов. Так, классы, использующие ленивую загрузку должны быть публичными, а их свойства должны иметь модификаторы **public** и **virtual**. Например, классы Player и Team могут иметь следующее определение:

```
public class Player
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Position { get; set; }
    public int Age { get; set; }

    public int? TeamId { get; set; }
    public virtual Team Team { get; set; }
}
public class Team
{
    public int Id { get; set; }
    public string Name { get; set; } // название команды
    public string Coach { get; set; } // тренер

    public virtual ICollection<Player> Players { get; set; }

    public Team()
    {
        Players = new List<Player>();
```

```
    }
}
```

В этом случае нам не потребуется использовать какие-то дополнительные методы, как Include или Load:

```
using (SoccerContext db = new SoccerContext())
{
    var players = db.Players.ToList();
    foreach (var p in players)
        Console.WriteLine($"{p.Name} - {p.Team.Name}");

    var teams = db.Teams.ToList();
    foreach (var t in teams)
    {
        Console.WriteLine($"{t.Name}");
        foreach (var p in t.Players)
            Console.WriteLine($"{p.Name}");
    }
}
```

Назад **Содержание** **Вперед**