

使用Entity Framework查询数据

北京理工大学计算机学院
金旭亮

EF数据查询基础

EF中编写数据查询代码的主要方式



示例：查询符合条件的单条记录

1 使用SingleOrDefault()扩展方法查找

```
// 查找ID=10的客户信息，找不到，返回 null  
var client = (from c in context.OrderClients  
              where c.ClientID == 10  
              select c).SingleOrDefault();
```

2 使用DbSet的Find()方法查找

```
// Find方法会先在内存中找，找不到之后再数据库中提取。  
// 注意：Find方法的参数是主键字段的某个值  
var c = context.OrderClients.Find(30);
```

注意

1

当使用标准LINQ查询扩展方法查询数据时，可能会有部分方法是无法使用的，比如Last()，其原因在于EF不知道如何把这些扩展方法转换为相应的SQL命令，会抛出NotSupportException。

2

另外，由于底层数据库不一样，因此，可能某扩展方法针对SQL Server可用，但针对其他类型的数据库（比如MySQL）就不行。

3

到底哪些方法可用哪些方法不可用，一个是看文档，更彻底的方法，是编写代码实地测试。

通知EF直接发送SQL命令

```
context.Database.ExecuteSqlCommand(  
    "delete from Client where ClientID={0}", ClientID);
```

这种方式简单直接，但要注意不要滥用，以避免将代码与底层数据库具体结构绑定得过于紧密。

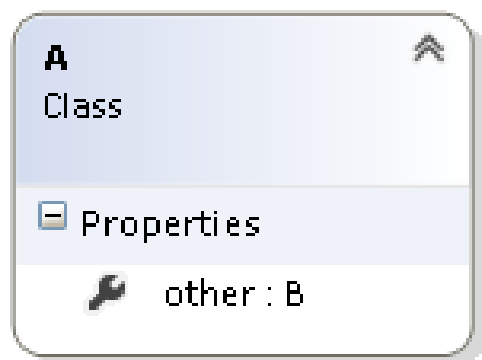
关联数据的查询

复习“面向对象编程基础知识”

C#实现对象之间的关联

```
class A
{
    public B other { get; set; }
}
```

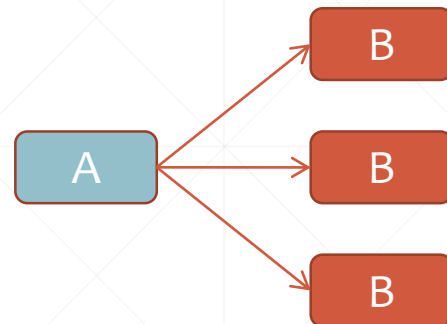
A对象与B对象之间，是
一对一关联



得到一个A对象，通过它的other属性，
就可以访问到一个B对象，我们就称A对
象可以“**导航**”到B对象，通常使用一
个箭头表示这种关联的“方向性”。

一对多关联

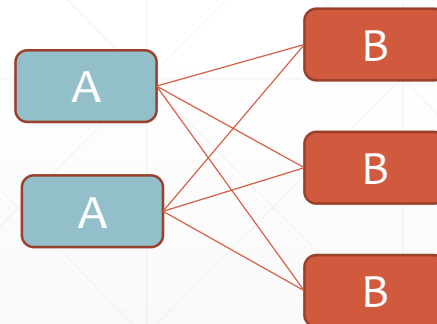
```
class A
{
    public List<B> others { get; set; }
}
```



多对多关联

```
class A
{
    public List<B> others { get; set; }
}

class B
{
    public List<A> others { get; set; }
}
```



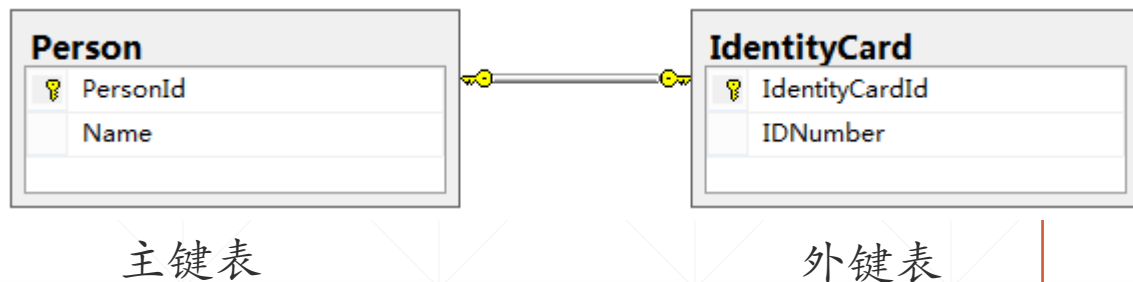
数据库表与Entity Framework数据实体间的关联

关系型数据库（比如SQL Server），可以在表之间直接建立**一对一**，**一对多**两种关联，但**不直接支持多对多关联**，多对多关联必须转换为两个一对多的关联。

Entity Framework中的数据实体类，由于使用面向对象编程语言实现，因此，**支持所有三种关联类型**。EF会在底层将其转换为数据库支持的两种关联类型。

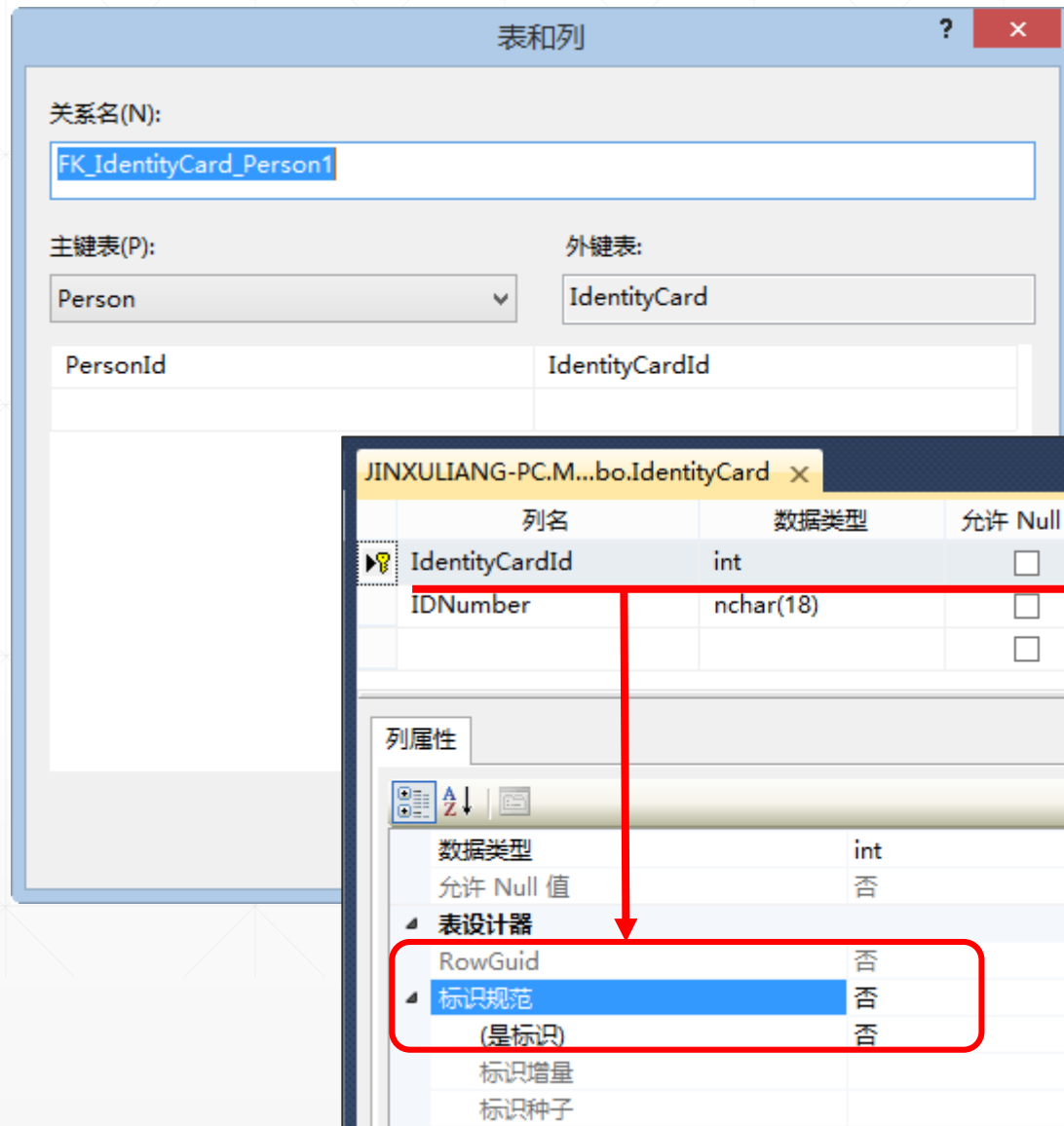
一对一关联实例

- 一个人只能有一个身份证号



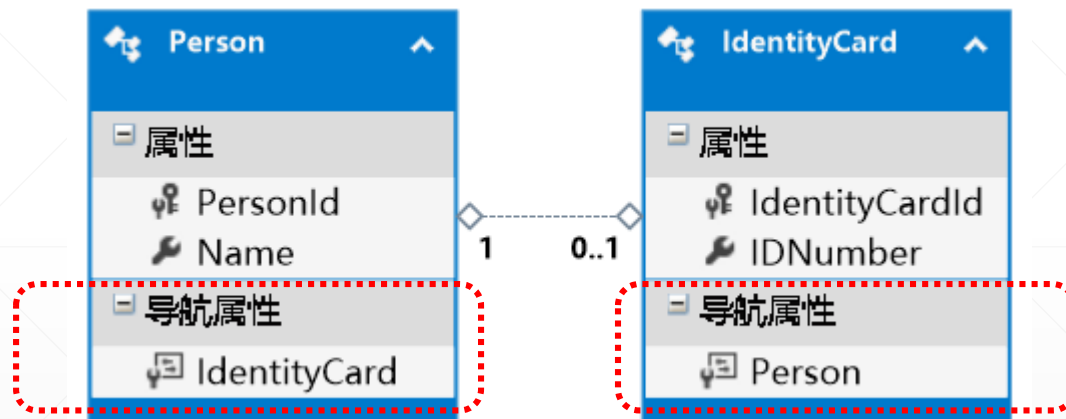
在SQL Server中，可以通过“**关系**”在两个表之间直接建立一对一的关联，请遵循以下设计规范：

- (1) 两个表都有主键
- (2) 外键表的主键不要设置为自增的标识字段。



Entity Framework中的一对一关联

使用Database First方式导入之后，EF将数据实体类Person与IdentityCard间的关联识别为“**1 对 0..1**”，但基本不会影响使用。



EF将引用实体对象的属性称为“**导航属性 (Navigation Properties)**”，默认设置下，当应用程序通过导航属性访问另一个数据实体（或实体集合）时，如果其数据还未装入，EF会向数据库发出SQL命令提取数据。

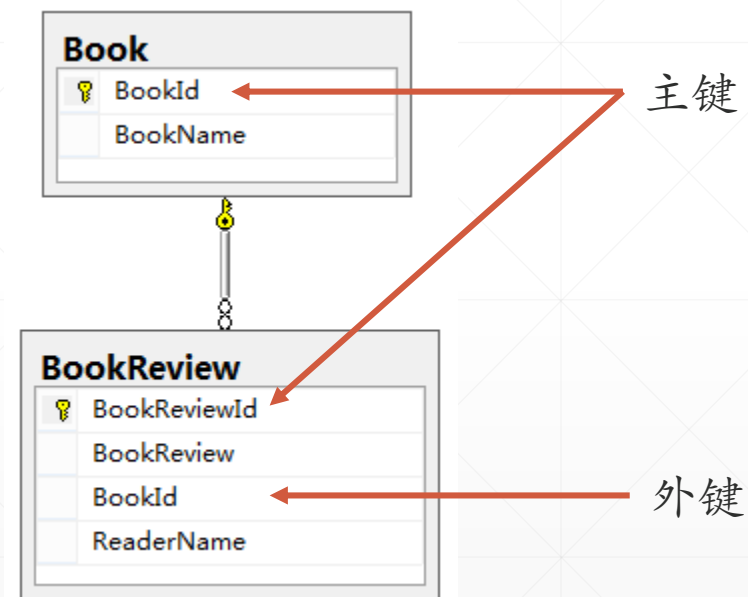
真的需要建立“一对一”关联？

一对一关联在实际开发中其实用处不大

- 具有一对一关联的表，可以被“合并”为一张大表，而关系数据库对属于一张表的字段的查询要比涉及多张一对一关联表的字段的查询效率要高。
- 如果在多个表间建立一对一的关联，实现CRUD时比较麻烦。

一对多关联

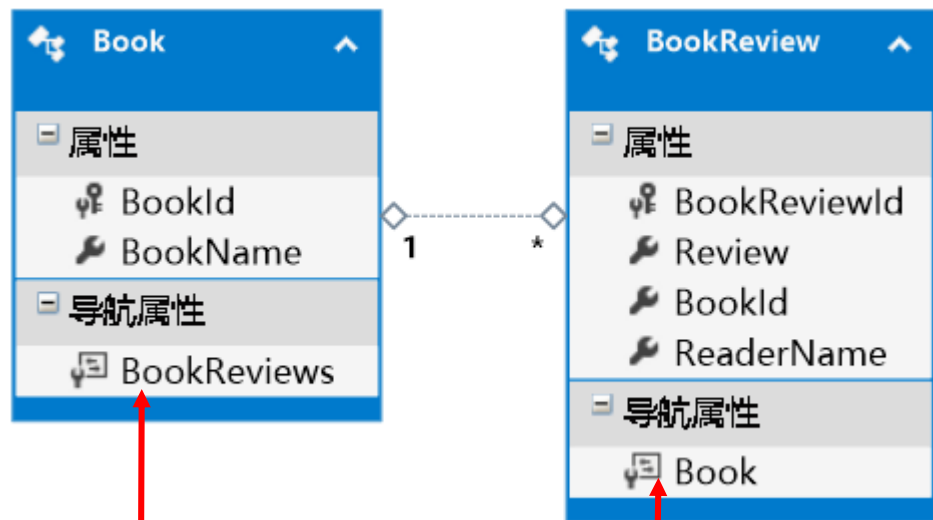
- 一本书可以有多个书评



在实际开发中，为简化开发，可为一对多关联设置“**级联删除**”特性。

标识	
(名称)	FK_BookReview_Book
说明	
数据库设计器	
INSERT 和 UPDATE 规则	
更新规则	不执行任何操作
删除规则	级联
强制外键约束	是
强制用于复制	是

Entity Framework中的一对多关联






```
public virtual Book Book { get; set; }
```

```
public virtual ICollection<BookReview> BookReviews { get; set; }
```

属性

▼ ↑ ✕

MyDBModel.FK_BookReview_Book Association ▼

End1 OnDelete

Cascade

End1 导航属性

BookReviews

End1 多重性

1 (一个 Book)

End1 角色名称

Book

End2 OnDelete

None

End2 导航属性

Book

End2 多重性

*** (BookReview 的集合)**

End2 角色名称

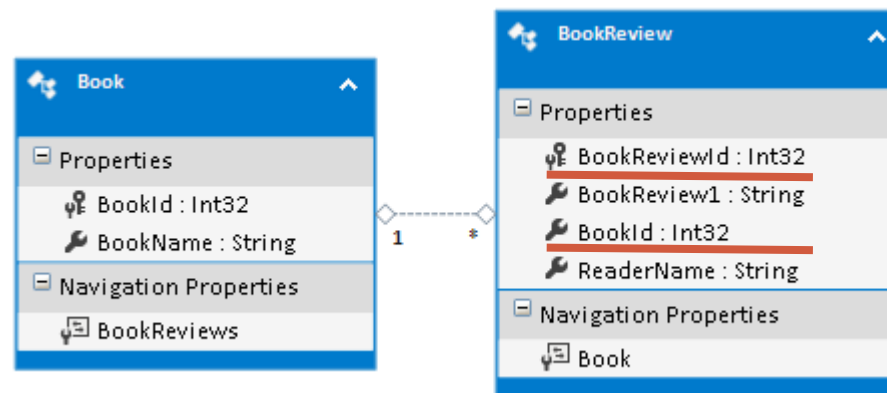
BookReview

名称

关联名称。

外键关联与独立关联

对于包容了主键（和外键）属性的一对多关联，位于“多”的一端的对象通过外键依赖于另一端，这种情形称为“**外键关联**（**foreign key association**）”。



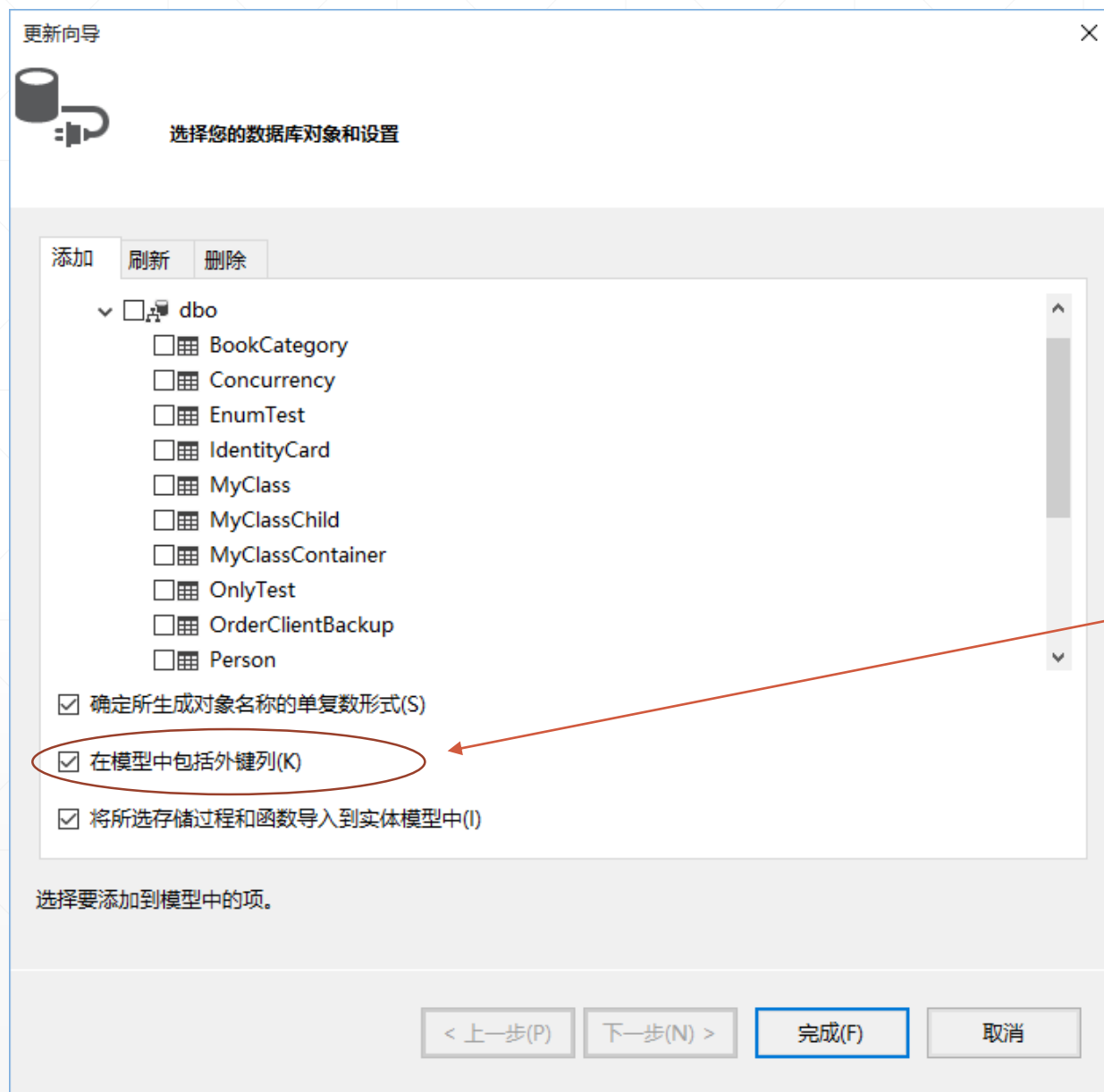
如果实体类中没有包容外键，则实体对象间只能通过导航属性相互关联，这种情形称为“**独立关联**（**independent association**）”。

最佳实践

强烈要求每个数据表都要定义主键，并且尽可能地采用“**自增**”等方式，让数据库自行生成主键字段值。

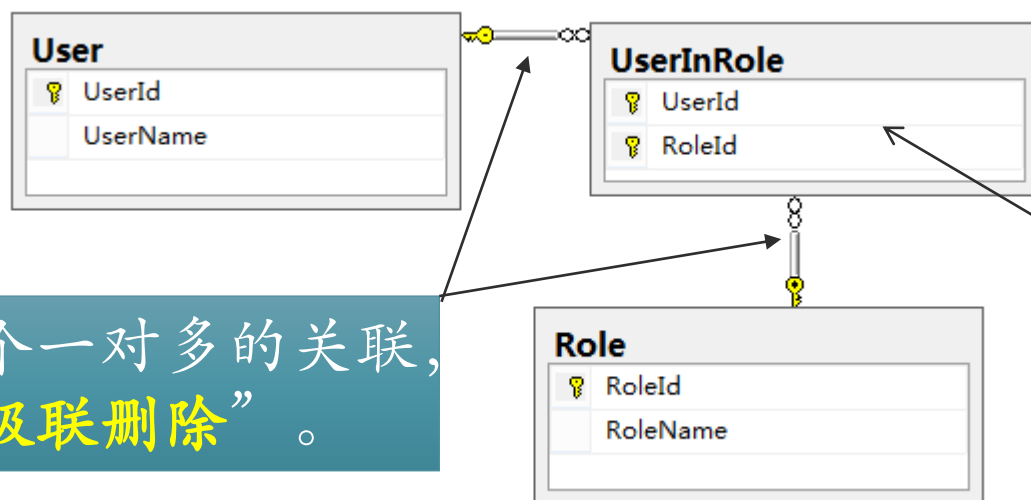
强烈建议在导入数据表时，选中这个选项，使用“外键关联”而不是“独立关联”。

在实际开发中，数据库越大，数据实体类也就越多，对象之间的关联关系也就越复杂，使用“外键关联”比“独立关联”更为灵活，也更易于维护。



多对多关联

- 一个**用户 (User)** 属于多种**角色 (Role)**
- 一种角色中又可以包容多个用户



为简化开发，两个一对多的关联，通常会设置为“**级联删除**”。

SQL Server不支持直接创建多对多的关联，需要将其拆分为两个一对多的关联。为此，需要引入**中间表**。

Entity Framework中的多对多关联

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

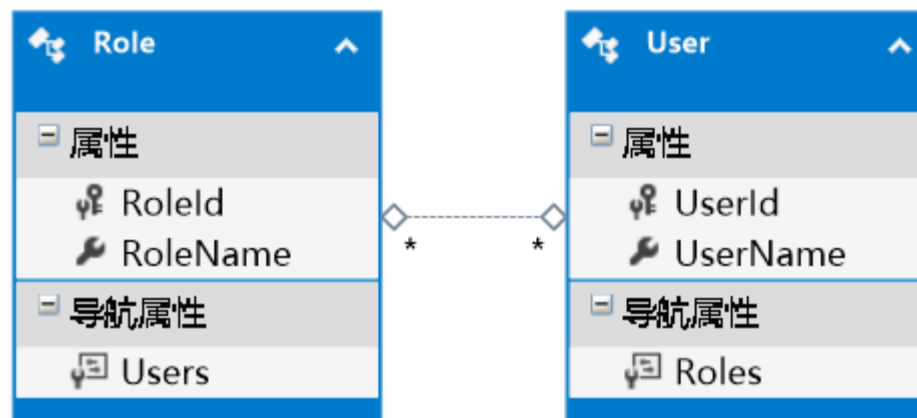
- ☐ IdentityCard
- ☐ OnlyTest
- ☐ OrderClient
- ☐ OrderClientBackup
- ☐ Person
- ☒ Role
- ☐ sysdiagrams
- ☒ User
- ☒ UserInRole

☐ Views

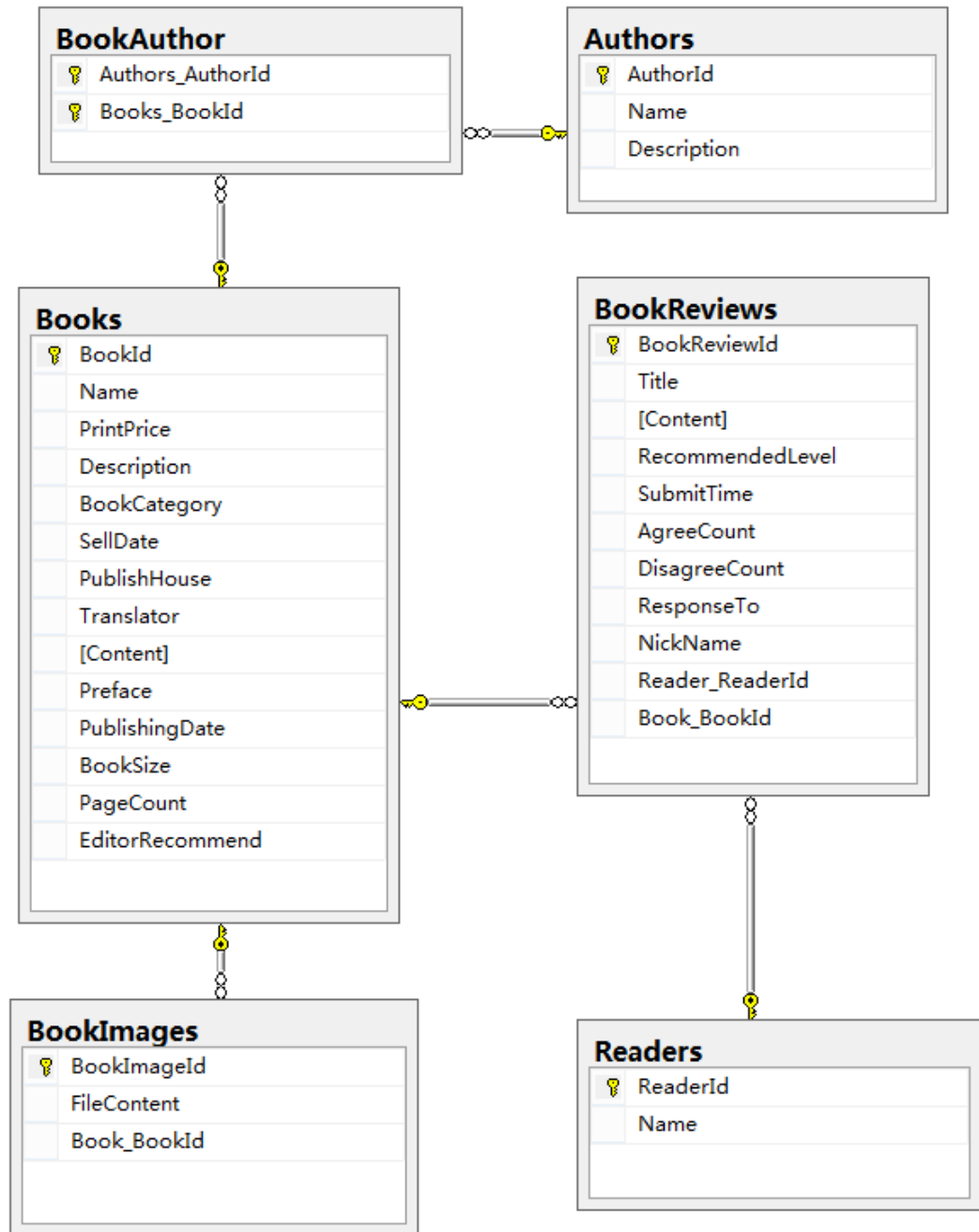
☐ Stored Procedures and Functions

如果中间表还包容有其它字段，则EF会将这三个表当成普通的两对“一对多”关联而引入。

选中三个表，如果中间表只包容两端两个对象的主/外键字段，则EF会帮助我们创建好两个对象之间的“多对多”关联



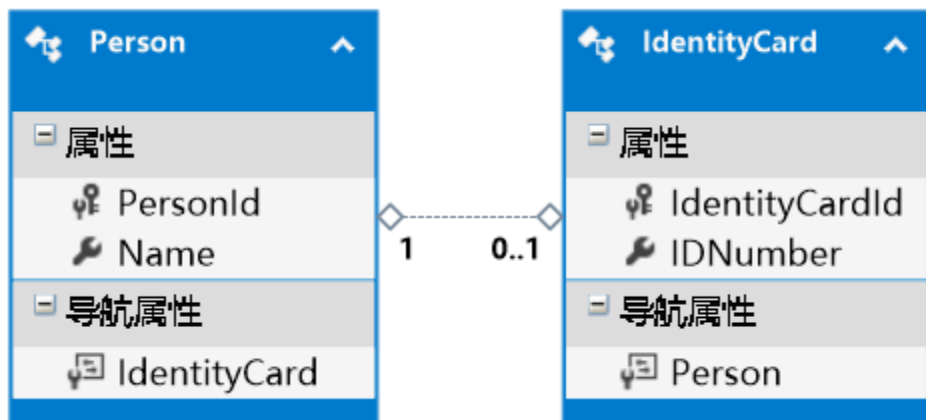
关联数据的加载



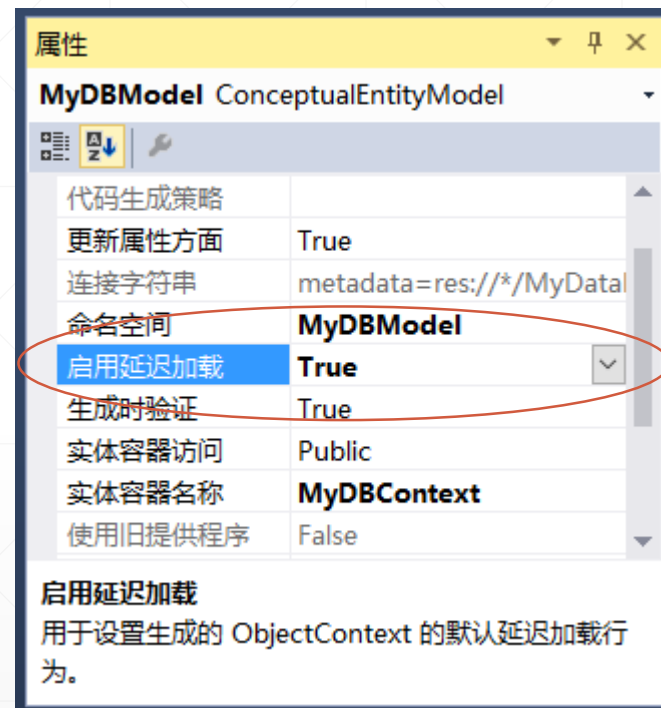
一个“电子书店”网站中与“书”相关的数据，体现为多级别的一对多关联……

那么，当提取一本书的信息时，相关联的数据是否也需要同时取出？

EF的默认策略：数据的分阶段提取



默认情况下，EF会启用延迟加载，因此，第一次仅提取Person表中的数据，当需要获取身份证信息时，再发出第二条SQL命令提取IdentityCard表中的数据



virtual关键字的作用

```
public partial class Person
{
    0 references
    public int PersonId { get; set; }
    2 references | 0/1 passing
    public string Name { get; set; }

    2 references | 0/2 passing
    public virtual IdentityCard IdentityCard { get; set; }
}
```

导航属性前有virtual属性，
当程序运行时，EF创建的实际上是一个“代理对象”



Watch 1		
Name	Value	Type
person	{System.Data.Entity.DynamicProxies.Person_D232541CB322F06A5E58CB3CD7}	OneToOne.Person {System.Data.Entity.DynamicProxi
[System.Data.Entity.I	{System.Data.Entity.DynamicProxies.Person_D232541CB322F06A5E58CB3CD7}	System.Data.Entity.DynamicProxies.Person_D232541C
IdentityCard	{System.Data.Entity.DynamicProxies.IdentityCard_A2A10F0A404C37C739BC3}	OneToOne.IdentityCard {System.Data.Entity.Dynamic
Name	"普通人9358"	string
PersonId	1	int

```

public partial class IdentityCard
{
    0 references
    public int IdentityCardId { get; set; }
    2 references | 0/1 passing
    public string IDNumber { get; set; }

    0 references
    public Person Person { get; set; }
}

```

```

public partial class Person
{
    0 references
    public int PersonId { get; set; }
    2 references | 0/1 passing
    public string Name { get; set; }

    2 references | 0/2 passing
    public IdentityCard IdentityCard { get; set; }
}

```

如果手工从数据实体类Person中移除virtual关键字，就失去了“动态加载关联数据”的功能

Watch 1		
Name	Value	Type
person	{OneToOne.Person}	OneToOne.Person
IdentityCard	null	OneToOne.IdentityCard
Name	"普通人9358"	string
PersonId	1	int

结论

默认情况下，EF为生成的数据实体类导航属性添加virtual关键字，在程序运行时，EF创建真实数据实体对象的一个“代理(proxy)”，此代理对象具有这样的特性：

只装载本数据实体直接管理的字段数据，而不加载关联对象的数据。仅当代码中通过本对象的导航属性访问另一数据实体对象时，EF才向数据库中发出新的SQL命令，动态地提取数据，创建一个数据实体对象返回给调用者。

这就是Entity Framework的数据“延迟加载 (Lazy Loading)”特性。

关于延迟加载.....

延迟加载的好处

- 每次加载的数据量小，提取数据速度快。

延迟加载的坏处

- 当一个数据实体包容相当多的导航属性时，会导致过多的数据库连接请求，给数据库服务器带来不小的压力，对网络应用程序的并发响应能力有不好的影响。

Eager Loading (预装载)

使用预先加载的方式，EF会生成一个连接查询，一次性地装载所有相关联的数据.....

```
[TestMethod]
public async Task TestFetchFirstPersonUseInclude()
{
    using (var context = new MyDBEntities())
    {
        Person person = await context.People
            .Include("IdentityCard").FirstOrDefaultAsync();
        if (person != null)
        {
            Console.WriteLine("姓名:{0} 身份证号:{1}",
                person.Name,
                person.IdentityCard.IDNumber);
        }
    }
}
```

```
file:///F:/EFQueryDemos/EntityAssociation/bin/Debug/EntityAssociation.EXE
Opened connection at 2015/12/4 15:39:02 +08:00

SELECT TOP (1)
    [Extent1].[PersonId] AS [PersonId],
    [Extent1].[Name] AS [Name],
    [Extent2].[IdentityCardId] AS [IdentityCardId],
    [Extent2].[IDNumber] AS [IDNumber]
FROM    [dbo].[Person] AS [Extent1]
LEFT OUTER JOIN [dbo].[IdentityCard] AS [Extent2] ON [Extent1].[PersonId] =
[Extent2].[IdentityCardId]

-- Executing at 2015/12/4 15:39:02 +08:00
-- Completed in 11 ms with result: SqlDataReader

Closed connection at 2015/12/4 15:39:02 +08:00
```

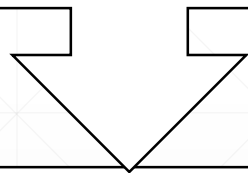
预装载数据的问题.....

1.构建出复杂的JOIN查询

2.加载的数据可能过多

开发建议

如果数据实体包容的导航属性较少，则使用默认的 Lazy Loading就行了，这适合于多数场景。



如果代码中需要访问同一个数据实体对象的多个导航属性，则可以提前使用Include()方法预加载关联数据，从而减少访问数据库的次数。

学习指南

数据查询是数据库应用程序中的核心功能，也是必须牢固掌握的，在看完所有视频之后，完成以下学习任务：

- 1 把所有示例认真地运行和阅读一遍，看懂每一行代码。
- 2 通过编写一些小的Demo，了解一下.NET所提供的集合标准查询方法哪些可以用于Entity Framework。
- 3 整理好实现各种典型数据查询功能的典型代码，开发备用。