

第一章

第二章 模型框架

一、 多对多关系处理（无有效载荷，即关联的表中无实际数据）



图1 多对多关系

如图 1 所示的多对多关系，在项目中添加实体后，形成如图 2 所示的模型结构。

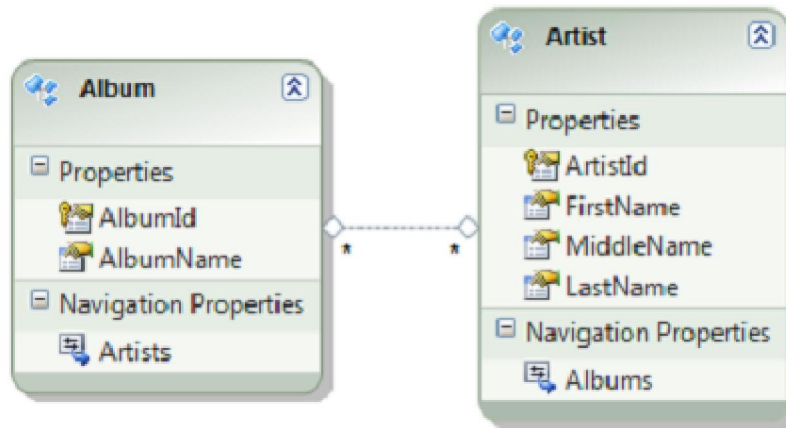


图2 多对多关系添加到 ado.net 模型中后的结果
处理代码如图 3 所示：

```

using (var context = new EFRecipesEntities())
{
    // add an artist with two albums
    var artist = new Artist { FirstName = "Alan", LastName = "Jackson" };
    var album1 = new Album { AlbumName = "Drive" };
    var album2 = new Album { AlbumName = "Live at Texas Stadium" };
    artist.Albums.Add(album1);
    artist.Albums.Add(album2);
    context.Artists.AddObject(artist);

    // add an album for two artists
    var artist1 = new Artist { FirstName = "Tobby", LastName = "Keith" };
    var artist2 = new Artist { FirstName = "Merle", LastName = "Haggard" };
    var album = new Album { AlbumName = "Honkytonk University" };
    artist1.Albums.Add(album);
    artist2.Albums.Add(album);
    context.Albums.AddObject(album);

    context.SaveChanges();
}

using (var context = new EFRecipesEntities())
{
    Console.WriteLine("Artists and their albums...");
    var artists = from a in context.Artists select a;
    foreach (var artist in artists)
    {
        Console.WriteLine("{0} {1}", artist.FirstName, artist.LastName);
        foreach (var album in artist.Albums)
        {
            Console.WriteLine("\t{0}", album.AlbumName);
        }
    }

    Console.WriteLine("\nAlbums and their artists...");
    var albums = from a in context.Albums select a;
    foreach (var album in albums)
    {
        Console.WriteLine("{0}", album.AlbumName);
        foreach (var artist in album.Artists)
        {
            Console.WriteLine("\t{0} {1}", artist.FirstName, artist.LastName);
        }
    }
}

```

图3 多对多关系对象处理代码

运行结果如图 4 所示

Artists and their albums...

Alan Jackson

Drive

Live at Texas Stadium

Tobby Keith

Honkytonk University

Merle Haggard

Honkytonk University

Albums and their artists...

Drive

Alan Jackson

Live at Texas Stadium

Alan Jackson

Honkytonk University

Tobby Keith

Merle Haggard

图4 多对多关系运行结果

二、 多对多关系处理（有有效载荷，即关联的表中有外键之外的实际数据）
如图 5 所示的多对多关系，其处理模型则如图 6 所示

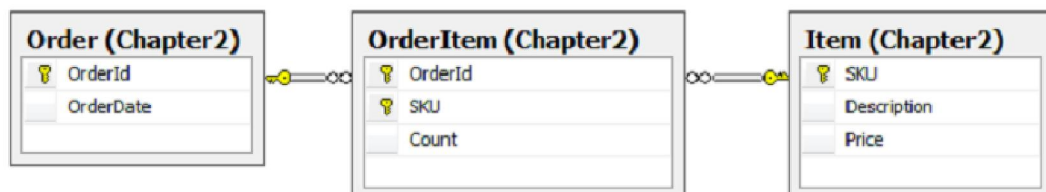


图5 多对多关系表结构

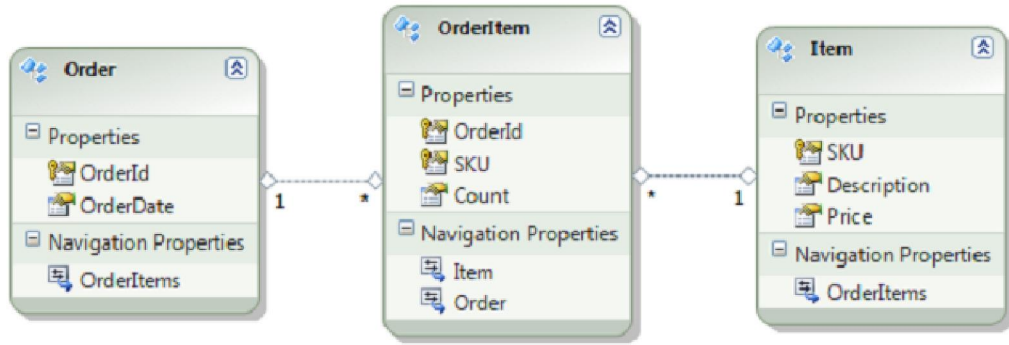


图6 多对多关系模型

处理代码如图 7 所示

```

using (var context = new EFRecipesEntities())
{
    var order = new Order { OrderId = 1,
        OrderDate = new DateTime(2010, 1, 18) };
    var item = new Item { SKU = 1729, Description = "Backpack",
        Price = 29.97M };
    var oi = new OrderItem { Order = order, Item = item, Count = 1 };
    item = new Item { SKU = 2929, Description = "Water Filter",
        Price = 13.97M };
    oi = new OrderItem { Order = order, Item = item, Count = 3 };
    item = new Item { SKU = 1847, Description = "Camp Stove",
        Price = 43.99M };
    oi = new OrderItem { Order = order, Item = item, Count = 1 };
    context.Orders.AddObject(order);
    context.SaveChanges();
}

using (var context = new EFRecipesEntities())
{
    foreach (var order in context.Orders)
    {
        Console.WriteLine("Order # {0}, ordered on {1}",
            order.OrderId.ToString(),
            order.OrderDate.ToShortDateString());
        Console.WriteLine("SKU\tDescription\tQty\tPrice");
        Console.WriteLine("----\t-----\t---\t-----");
        foreach (var oi in order.OrderItems)
        {
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", oi.Item.SKU,
                oi.Item.Description, oi.Count.ToString(),
                oi.Item.Price.ToString("C"));
        }
    }
}
    
```

图7 多对多关系处理代码

处理结果如所图 8 示

Order # 1, ordered on 1/18/2010

SKU	Description	Qty	Price
---	-----	---	-----
1729	Backpack	1	\$29.97
1847	Camp Stove	1	\$43.99
2929	Water Filter	3	\$13.97

图8 多对多关系处理结果

三、 自引用数据表处理

如图 9 所示的自引用关系，其模型如图 10 所示。

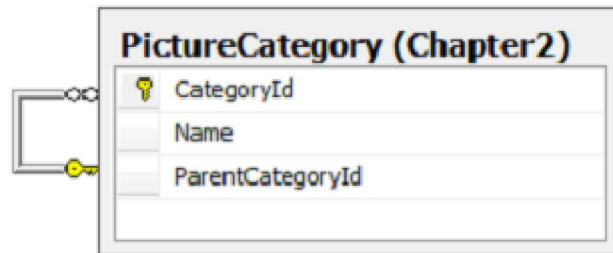


图9 自引用关系

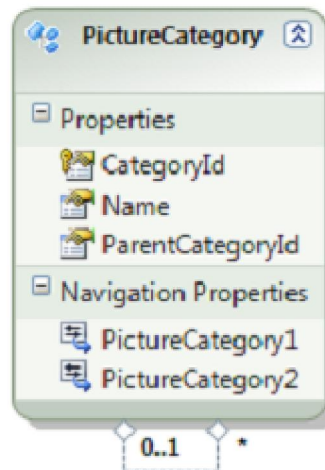


图10 自引用模型

模型中，有两个导航属性，其名称为：PictureCategory1 和 PictureCategory2，这是因为这两个导航属性分别对应父表和子表的对应记录引用。为了明确父表和子表对应的实际关系，分别修改对应的属性名称，以实现规范化和方便代码编写，找到代表 0..1 端的导航属性，由于其代表着父表记录，修改其属性名称为：ParentCategory，找到代表*端的导航属性，由于其代表着子表记录，修改其属性名称为：SubCategory，修改后的模型如图 11 所示。

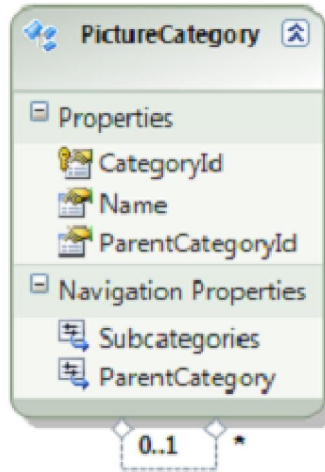


图11 修改导航模型名称

其处理代码为:

```
static void RunExample()
{
    using (var context = new EFRecipesEntities())
    {
        var louvre = new PictureCategory { Name = "Louvre" };
        var child = new PictureCategory { Name = "Egyptian Antiquites" };
        louvre.Subcategories.Add(child);
        child = new PictureCategory { Name = "Sculptures" };
        louvre.Subcategories.Add(child);
        child = new PictureCategory { Name = "Paintings" };
        louvre.Subcategories.Add(child);
        var paris = new PictureCategory { Name = "Paris" };
        paris.Subcategories.Add(louvre);
        var vacation = new PictureCategory { Name = "Summer Vacation" };
        vacation.Subcategories.Add(paris);
        context.PictureCategories.AddObject(paris);
        context.SaveChanges();
    }

    using (var context = new EFRecipesEntities())
    {
        PictureCategory root = (from c in context.PictureCategories
                               where c.ParentCategory == null
                               select c).FirstOrDefault();

        Print(root, 0);
    }
}

static void Print(PictureCategory cat, int level)
```

```

{
    StringBuilder sb = new StringBuilder();
    Console.WriteLine("{0}{1}", sb.Append(' ', level).ToString(), cat.Name);
    foreach (PictureCategory child in cat.Subcategories)
    {
        Print(child, level + 1);
    }
}

```

运行结果为:

Summer Vacation

Paris

Louvre

Egyptian Antiquites

Sculptures

Paintings

四、 实体数据分散在多个表

如果多个表中的记录通过共用同一个主键值, 因此而希望把这多个表中的数据整合到相同的实体中去, 则可采用以下方案:

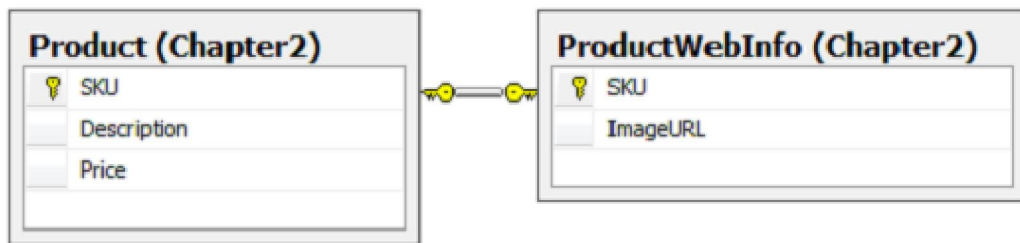


图12 相同主键的多个表结构

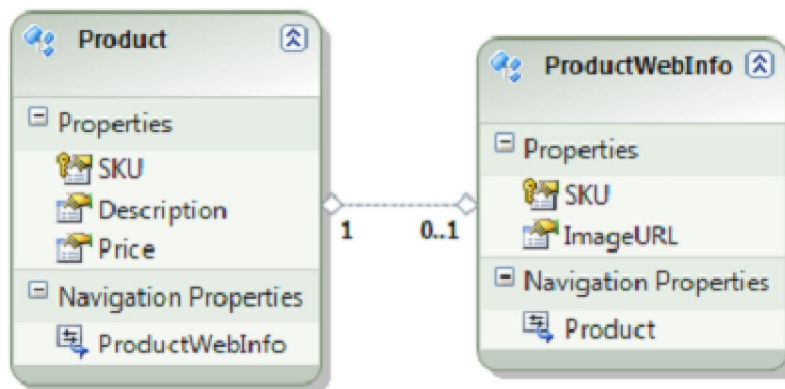


图13 向导创建的不同主键的多个表结构模型

如图 12 所示的多个表中记录通过相同的主键分别存储同一实体的一部分数据, 通过向导添加这两个表后, 其模型结构图如图 13 所示。

向导创建的模型仍是把同一实体的数据分散在两个表中, 需要调整模型以实现把同一实体的所有数据存储在同一实体中, 以简化代码编写, 调整步骤为:

1. 通过“复制/粘贴”操作, 把实体 ProductWebInfo 中的属性 ImageURL 复制到 Product

中，但不要复制 ProductWebInfo 中的属性 SKU。这么做的原因是因为想把 ProductWebInfo 表中数据的处理合并到表 Product 中数据过程中。

2. 右击实体“ProductWebInfo”，选择“删除”，在弹出的如图 14 所示对话框中，选择“否”，以保留 ProductWebInfo 的定义在模型层。

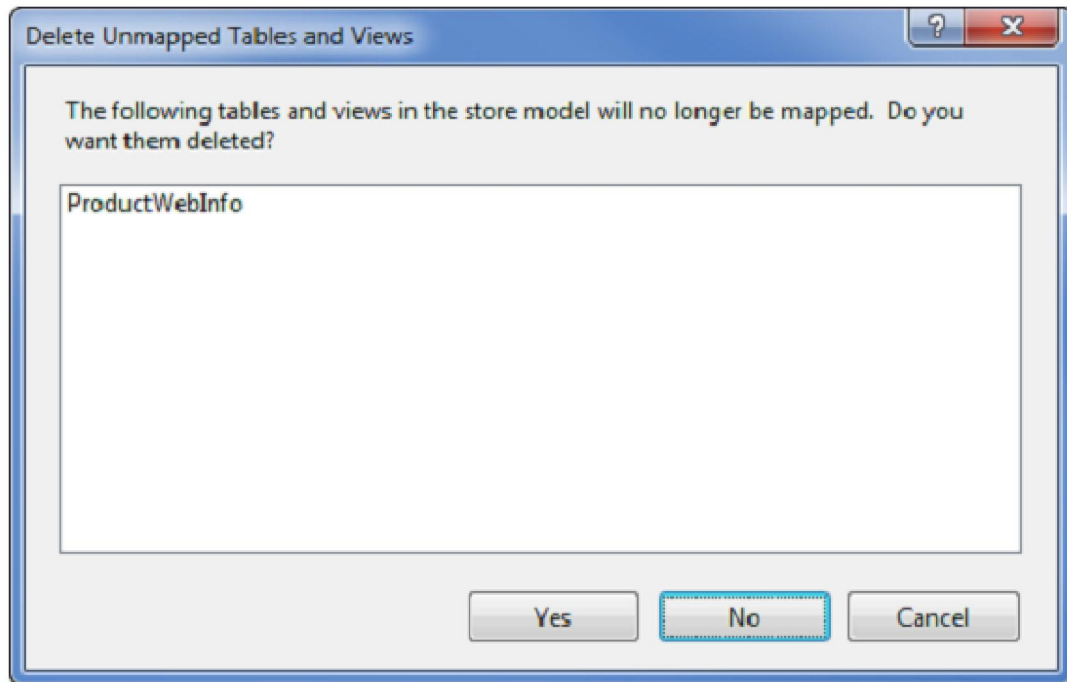


图14 删除实体 ProductWebInfo

3. 单击“Product”实体，打开“映射详细信息视图”。
4. 在 Product 实体的 Mapping Details 视图中，单击“添加表或视图”，并选择 ProductWebInfo 表，此操作添加实体 Product 到表 ProductWebInfo 的映射关系。
5. 在 Mapping Detail 视图的 ProductWebInfo 表中，设置 ImageURL 属性映射到 ImageURL 列；同样的，添加 SKU 属性映射到表的 SKU 列，修改结果如图图 15 所示。

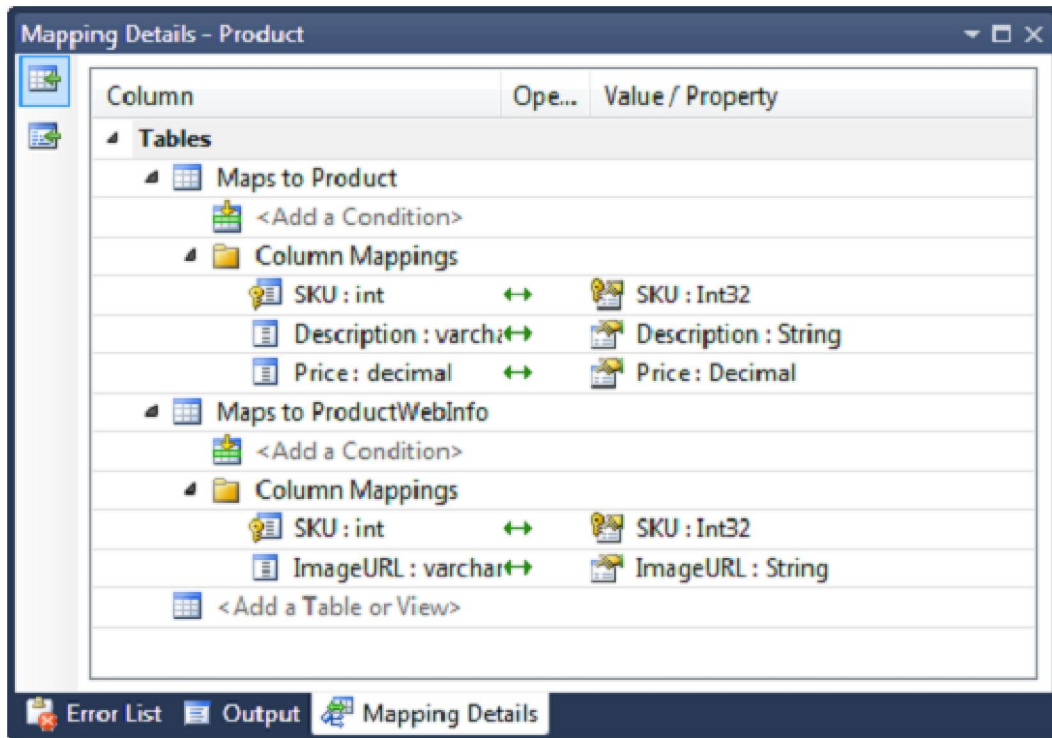


图15 添加映射关系后的结果

添加数据到两个表的操作，只需要按一个实体中的数据处理即可，代码如下所示：

```
using (var context = new EFRecipesEntities())
{
    var product = new Product { SKU = 147,
                               Description = "Expandable Hydration Pack",
                               Price = 19.97M, ImageURL = "/pack147.jpg" };
    context.Products.AddObject(product);
    product = new Product { SKU = 178,
                            Description = "Rugged Ranger Duffel Bag",
                            Price = 39.97M, ImageURL = "/pack178.jpg" };
    context.Products.AddObject(product);
    product = new Product { SKU = 186,
                            Description = "Range Field Pack",
                            Price = 98.97M, ImageURL = "/noimage.jp" };
    context.Products.AddObject(product);
    product = new Product { SKU = 202,
                            Description = "Small Deployment Back Pack",
                            Price = 29.97M, ImageURL = "/pack202.jpg" };
    context.Products.AddObject(product);

    context.SaveChanges();
}

using (var context = new EFRecipesEntities())
```

```

{
    foreach (var p in context.Products)
    {
        Console.WriteLine("{0} {1} {2} {3}", p.SKU, p.Description,
            p.Price.ToString("C"), p.ImageURL);
    }
}

```

执行结果为：

```

147 Expandable Hydration Pack $19.97 /pack147.jpg
178 Rugged Ranger Duffel Bag $39.97 /pack178.jpg
186 Range Field Pack $98.97 /noimage.jpg
202 Small Deployment Back Pack $29.97 /pack202.jpg

```

五、 一个表的数据分割到多个实体中

如果一个表中的部分列数据需要经常访问，而部分列数据很大却很少访问。为了提高系统性能，需要避免每一次访问数据时都加载那些代价很大的字段，实现方法则可以通过把这个表分割到两个或多个实体中。

如图 16 所示的表结构，其中的字段：Title 和 ThumbnailBits 字段基本上每次访问都需要处理，而 HighResolutionBits 字段则很少使用到，因此，为了提高系统性能，计划把此表分割到两个实体中。

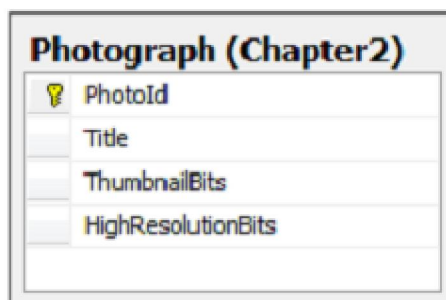


图16 需要进行分割的表结构

把表的字段分割到两个实体的操作过程为：

1. 通过向导，添加表 Photograph 到实体模型中。
2. 添加一个新的实体 PhotographFullImage 到模型中，并且复制 Photograph 实体中的属性 PhotoId 到实体 PhotographFullImage 中。
3. 通过“剪切/粘贴”操作，把实体 Photograph 中的属性 HighResolutionBits 移到实体 PhotographFullImage 中。
4. 打开 PhotographFullImage 实体的“Mapping Detail 视图”，添加表 Photograph 到映射表中，然后映射实体中的属性 PhotoId 和 HighResolutionBits 到 photograph 表对应字段中。处理结果如图 17 所示。

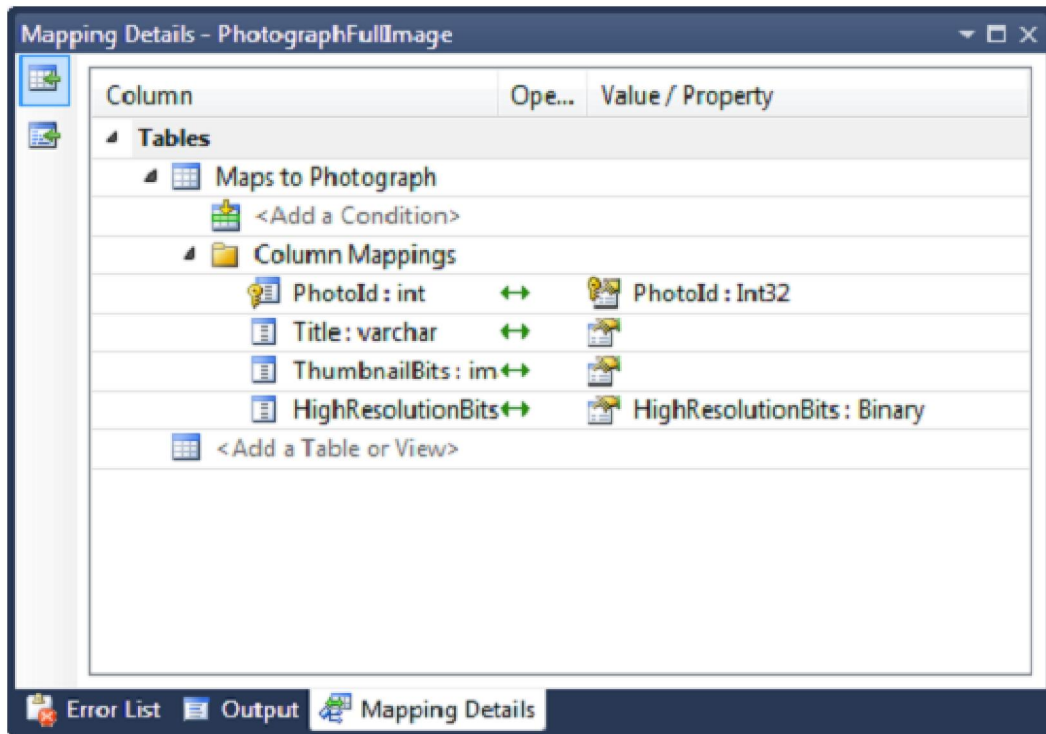


图17 修改后的实体 PhotographFullImage

5. 右击实体 Photograph，选择“添加->关联”，在实体 Photograph 和 PhotographFullImage 实体之间添加“一对一”关联，如图 18 所示。注意要确保两点：关联两端的类型都是“1”类型；在实体 PhotographFullImage 中取消“添加外键属性”复选框。

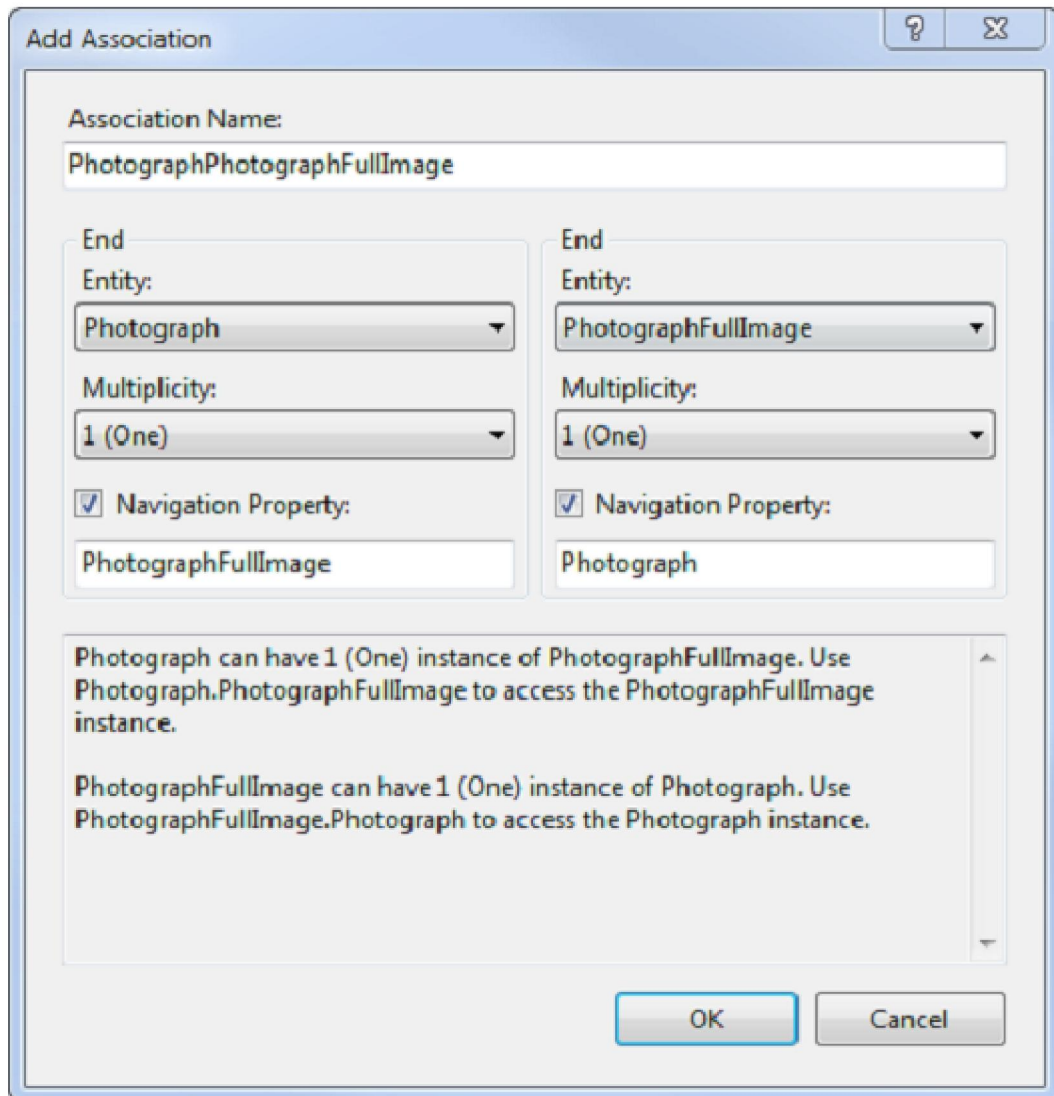


图18 添加实体间“一对一”关联

6. 右击添加了“一对一”关联，单击“...”按钮，设置 Photograph 表为主表，Photold 为主表和从表中的键，操作结果如图 19 所示。

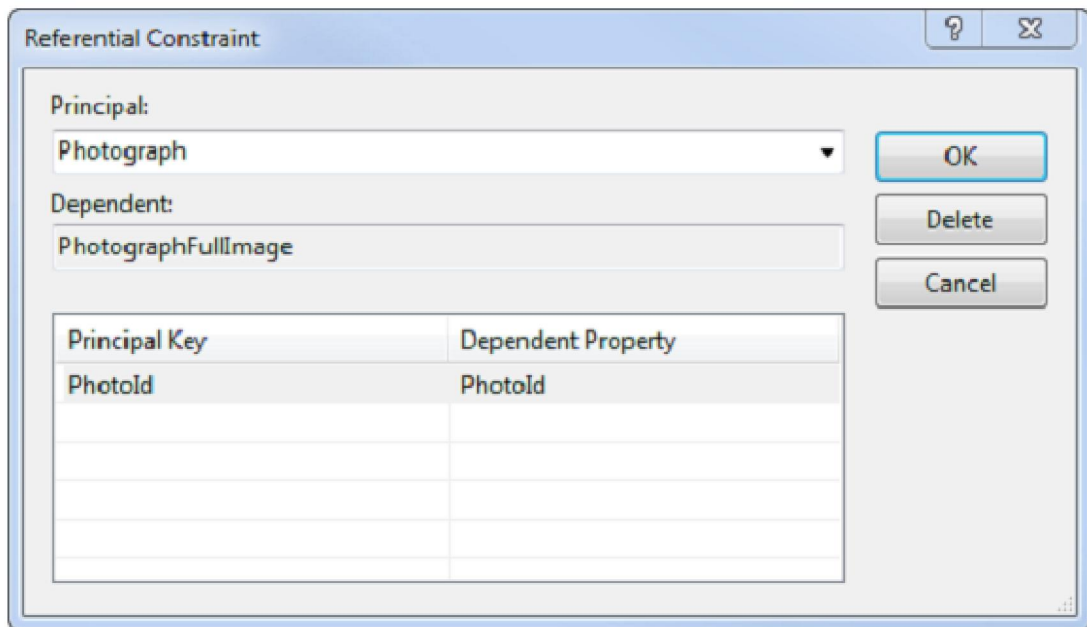


图19 添加关联的约束

实体模型处理结果如图 20 所示。

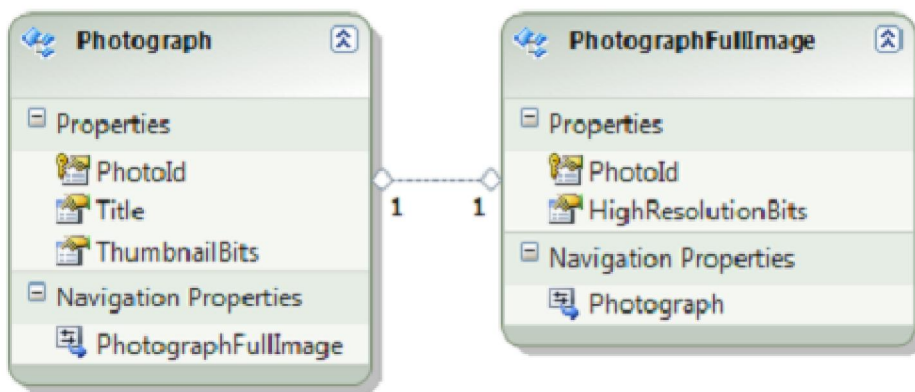


图20 模型处理结果

表中数据分到两个实体中的处理代码为：

```

byte[] thumbBits = new byte[100];
byte[] fullBits = new byte[2000];
using (var context = new EFRecipesEntities())
{
    var photo = new Photograph { PhotoId = 1, Title = "My Dog",
                               ThumbnailBits = thumbBits };
    var fullImage = new PhotographFullImage { PhotoId = 1,
                                              HighResolutionBits = fullBits };
    photo.PhotographFullImage = fullImage;
    context.Photographs.AddObject(photo);
    context.SaveChanges();
}

```

```

using (var context = new EFRecipesEntities())
{
    foreach (var photo in context.Photographs)
    {
        Console.WriteLine("Photo: {0}, ThumbnailSize {1} bytes",
            photo.Title, photo.ThumbnailBits.Length.ToString());

        // explicitly load the "expensive" entity, PhotographFullImage
        photo.PhotographFullImageReference.Load();
        Console.WriteLine("Full Image Size: {0} bytes",
            photo.PhotographFullImage.HighResolutionBits.Length.ToString());
    }
}

```

执行结果为:

Photo: My Dog, Thumbnail Size: 100 bytes

Full Image Size: 2000 bytes

六、 每个表对应派生结构中的一个实体

如图 21 所示，Business 表中存储基本信息，eCommerce 和 Retail 存储补充信息，表之间的关系是 1 对 0.1，需要针对三个表分别设计对应的实体类。

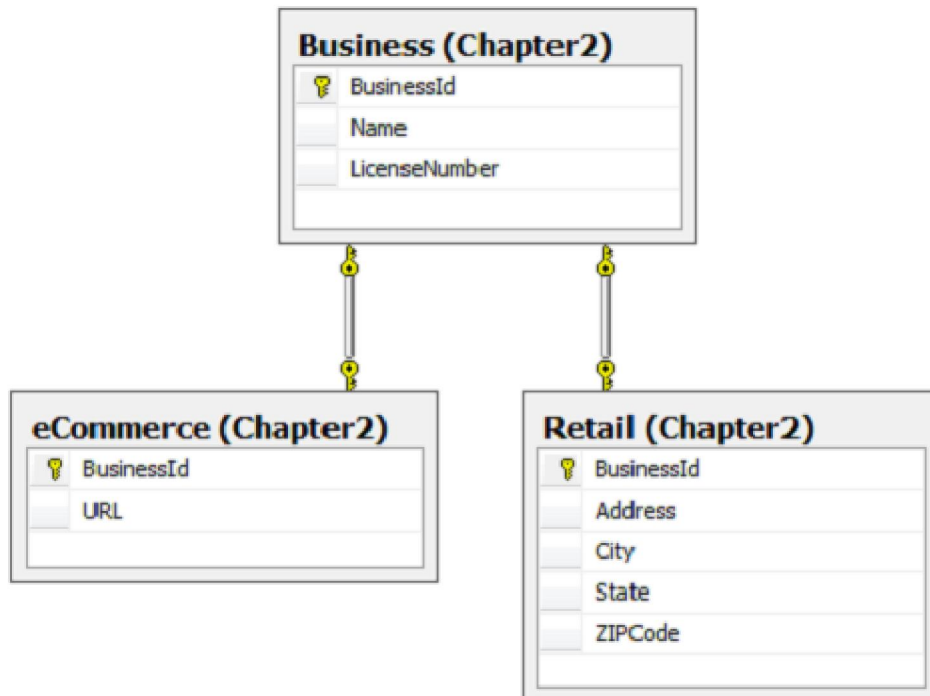


图21 表中存储补充信息

创建实体模型的操作为:

1. 通过向导添加三个表到模型中，保留其中的外键属性。
2. 删除实体 eCommerce 和 Business 之间的关系，删除实体 Retail 和 Business 之间的关系。

3. 右击“Business”实体，选择“继承”，在弹出的对话框中选择 Business 为基类，Retail 为派生类；用相同的方法添加 eCommerce 和 Business 之间的继承关系，操作对话框如所示。

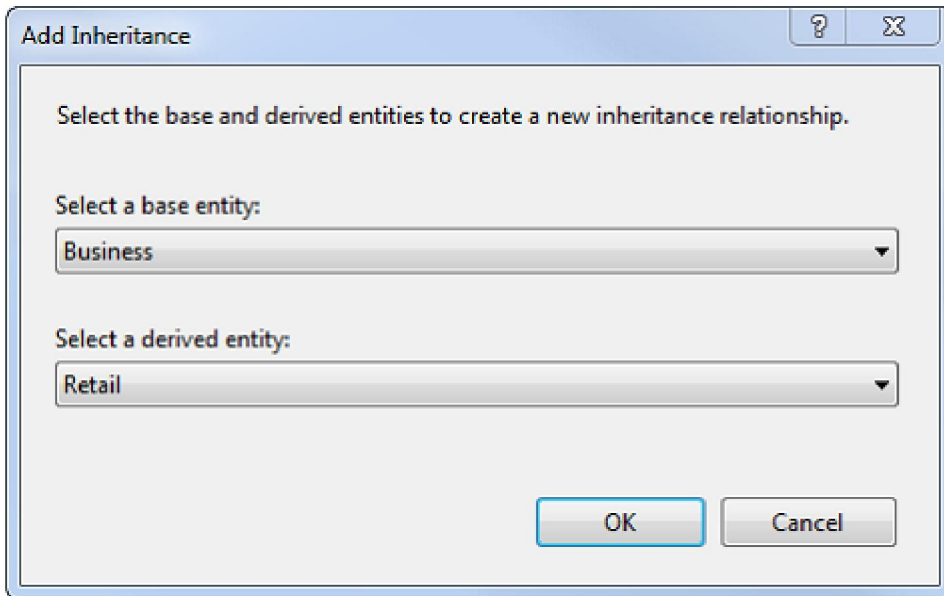


图22 添加继承关系

4. 删除 eCommerce 和 Retail 中的 BusinessId 属性，此属性从基类中获取。
5. 在 eCommerce 和 Retail 的“mapping detail 视图”中，设置 BusinessId 属性映射到表中的 BusinessId 字段，如图 23 所示，实体模型最后调整结果如图 24 所示。

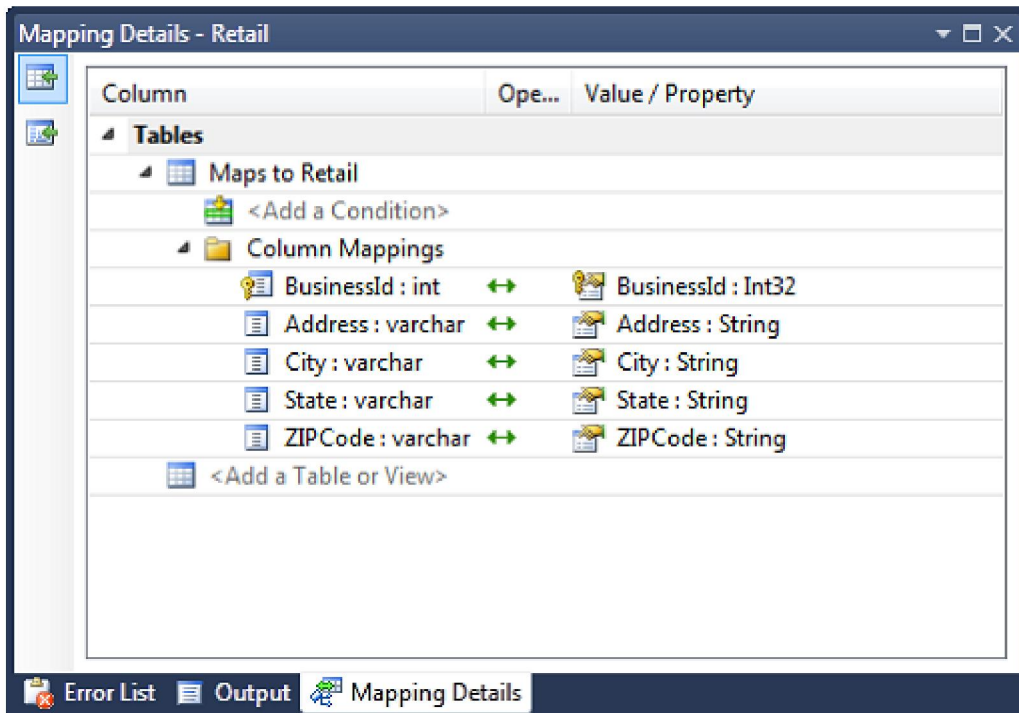


图23 添加 BusinessId 属性映射关系

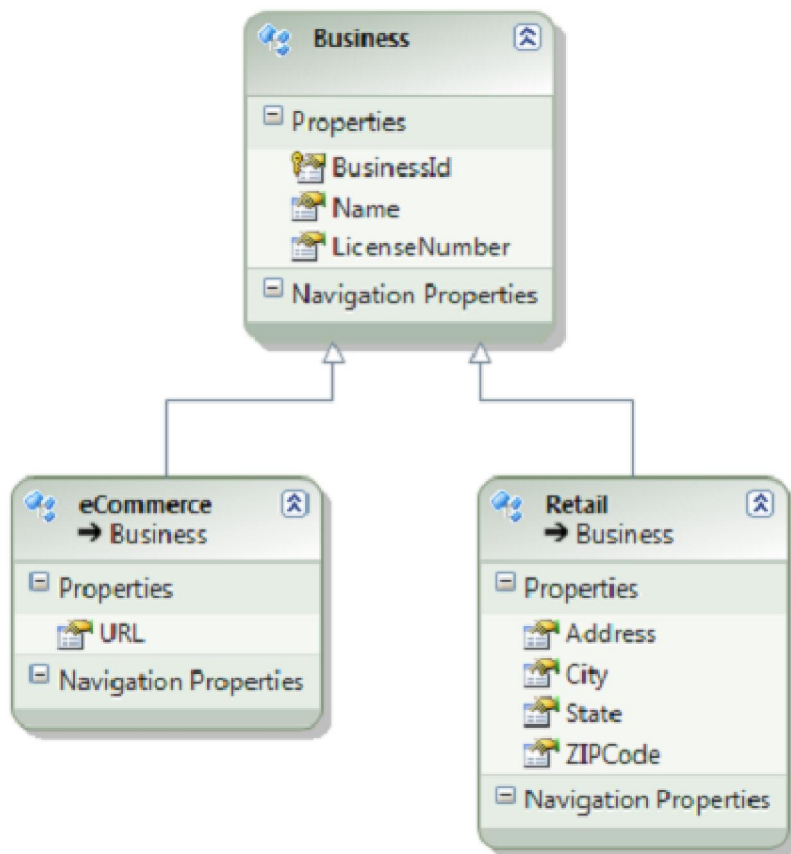


图24 实体模型调整结果

处理代码为:

```

using (var context = new EFRecipesEntities())
{
    var business = new Business { Name = "Corner Dry Cleaning",
                                  LicenseNumber = "100x1" };
    context.Businesses.AddObject(business);
    var retail = new Retail { Name = "Shop and Save", LicenseNumber = "200C",
                              Address = "101 Main", City = "Anytown",
                              State = "TX", ZIPCode = "76106" };
    context.Businesses.AddObject(retail);
    var web = new eCommerce { Name = "BuyNow.com", LicenseNumber = "300AB",
                              URL = "www.buynow.com" };
    context.Businesses.AddObject(web);
    context.SaveChanges();
}

using (var context = new EFRecipesEntities())
{
    Console.WriteLine("\n--- All Businesses ---");
}
  
```



```

foreach (var b in context.Businesses)
{
    Console.WriteLine("{0} ({1})", b.Name, b.LicenseNumber);
}

Console.WriteLine("\n--- Retail Businesses ---");
foreach (var r in context.Businesses.OfType<Retail>())
{
    Console.WriteLine("{0} ({1})", r.Name, r.LicenseNumber);
    Console.WriteLine("{0}", r.Address);
    Console.WriteLine("{0}, {1} {2}", r.City, r.State, r.ZIPCode);
}

Console.WriteLine("\n--- eCommerce Businesses ---");
foreach (var e in context.Businesses.OfType<eCommerce>())
{
    Console.WriteLine("{0} ({1})", e.Name, e.LicenseNumber);
    Console.WriteLine("Online address is: {0}", e.URL);
}
}

```

执行结果为:

```

--- All Businesses ---
Corner Dry Cleaning (#100X1)
Shop and Save (#200C)
BuyNow.com (#300AB)

--- Retail Businesses ---
Shop and Save (#200C)
101 Main
Anytown, TX 76106

---- eCommerce Businesses ---
BuyNow.com (#300AB)
Online address is: www.buynow.com

```

其中:

子类对象（包括 `eCommerce` 和 `Retail` 类的所有实例），都可以通过 `Business` 类完成操作。在查询操作端，可以在 `Business` 实体集中重复操作，以访问所有 `Business` 表中的所有记录，对于派生类数据，即 `eCommerce` 表和 `Retail` 表中的数据，可以通过 `OfType<>()` 方法来指定派生类型进行过滤操作，得到对应表中的数据，具体代码如下：

```

foreach (var r in context.Businesses.OfType<Retail>())
{
    .....
}

```

七、使用条件过滤对象集

通过持久的过滤条件过滤表中数据，得到表数据子集。

如图 25 所示，Account 表中，包括所有记录，其中，如果字段 DeletedOn 的值不为 null，而是记录了删除账号的日期，则表明对应账号已失效，对于有效账号，则其值为 null，为了方便访问其中的全部有效账号或无效账号，可以设计持久过滤条件实现。

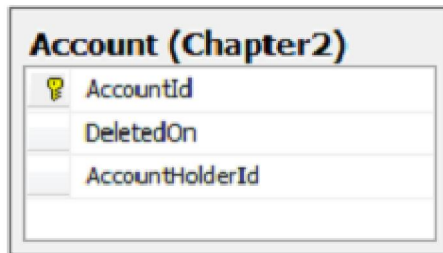


图25 Account 表结构

1. 通过向导，添加表 Account 到实体模型中。
2. 打开 Account 表的映射详细信息视图，单击添加“条件”，并选择 DeletedOn 列，在操作列中，选择“IS”，在“值/属性”列中，选择“Null”，此操作创建“当 DeletedOn 列的值为空的一个映射条件”。操作界面如图 26 所示。

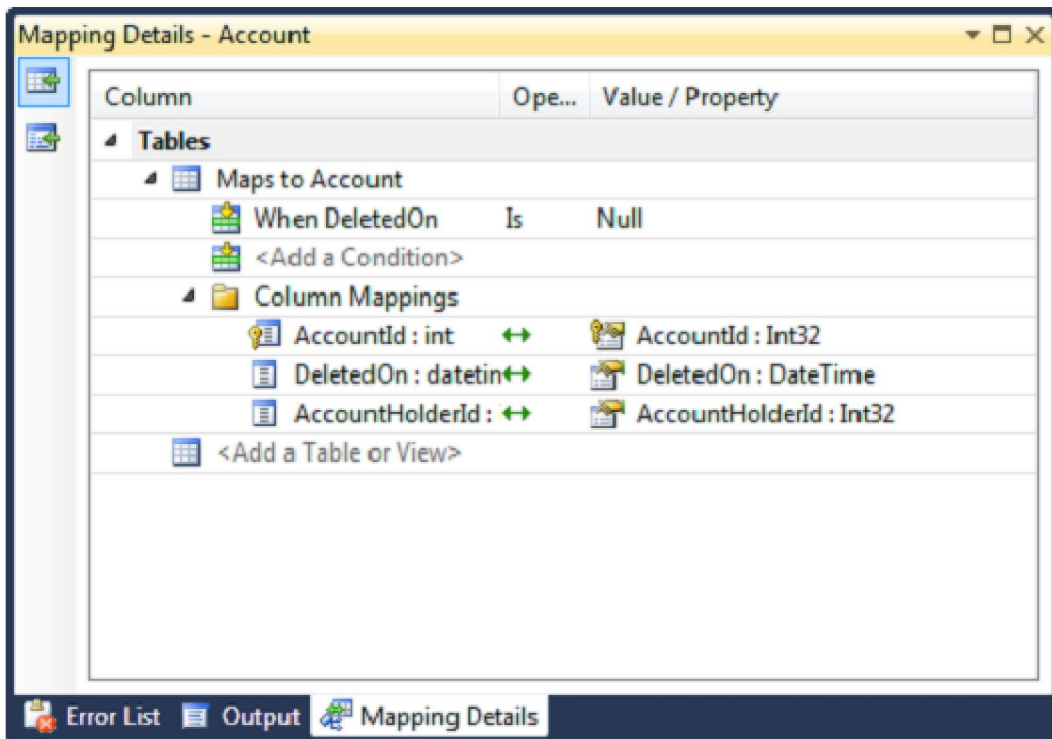


图26 添加映射条件

处理代码为：

```
using (var context = new EFRecipesEntities())
{
    context.ExecuteStoreCommand(@"insert into chapter2.account
        (DeletedOn,AccountHolderId) values ('2/10/2009',1728)");
}
```

```

var account = new Account { AccountHolderId = 2320 };
context.Accounts.AddObject(account);
account = new Account { AccountHolderId = 2502 };
context.Accounts.AddObject(account);
account = new Account { AccountHolderId = 2603 };
context.Accounts.AddObject(account);
context.SaveChanges();
}

using (var context = new EFRecipesEntities())
{
    foreach (var account in context.Accounts)
    {
        Console.WriteLine("Account Id = {0}",
            account.AccountHolderId.ToString());
    }
}

```

其中，通过 `ExecuteStoreCommand()`方法添加记录，但是由于此记录的 `DeletedOn` 字段值不为 `null`，所以，其后的查询操作没有访问到此记录，因为通过过滤条件，此记录不在正常查询得到的子集中。

执行结果为：

Account Id = 2320

Account Id = 2502

Account Id = 2603

但是，如果需要得到此记录，又该如何实现呢？

八、 Modeling Table pre Hierarchy Inheritance(P89)

问题：

对于在一个表中，通过列的值来区分表中各行数据代表不同类型的数据，同时还使用类的派生结构来实现。

解决方案：

如果有如图 27 所示的表结构，`Employee` 表中包含有按小时计费的员工和全职员工，`EmployeeType` 列是用于区分行数据具体是哪一种员工的列，当此列值是 1 时，行数据对应全职员工，当此列值是 2，行数据对应按小时计费的员工


Employee (Chapter2)	
	EmployeeId
	EmployeeType
	FirstName
	LastName
	Salary
	Wage

图27 Employee 表包含有全职员工和按小时计费的两种雇员

实现步骤:

1. 通过向导, 把表添加到实体模型中。
2. 添加实体到实体模型中, 命名为 FullTimeEmployee, 并选择 Employee 为基类, 同样添加新实体 HourlyEmployee, 基类为 Employee。
3. 把 Employee 类中的 Salary 属性移动到 FullTimeEmployee 中, 把 Wage 属性移动到 HourlyEmployee 中, 因为 Salary 只在 FullTimeEmployee 员工中才有, 而 Wage 只在小时工中才有。
4. 点击 FullTimeEmployee 实体, 查看其映射详细信息, 添加 Employee 表到 FullTimeEmployee 的映射表中, 确定 Salary 属性是映射到 Salary 列。
5. 在映射详细信息窗口中, 添加 EmployeeType 列添加条件, 设置运算符为 “=”, 值为 1, 结果如图 28 所示。
6. 如 4、5 所示方法, 为 HourEmployee 实体设置映射表, 添加条件为 EmployeeType 列的值为 2。
7. 右击 Employee 实体, 并选择属性, 修改其 “抽象” 属性值为 “TRUE”, 以此 Employee 设置成为抽象类, 这样, 所有 Employee 只能是 HourlyEmployee 或 FullTimeEmployee 中的一种。
8. 删除 Employee 实体中的 EmployeeType 属性。

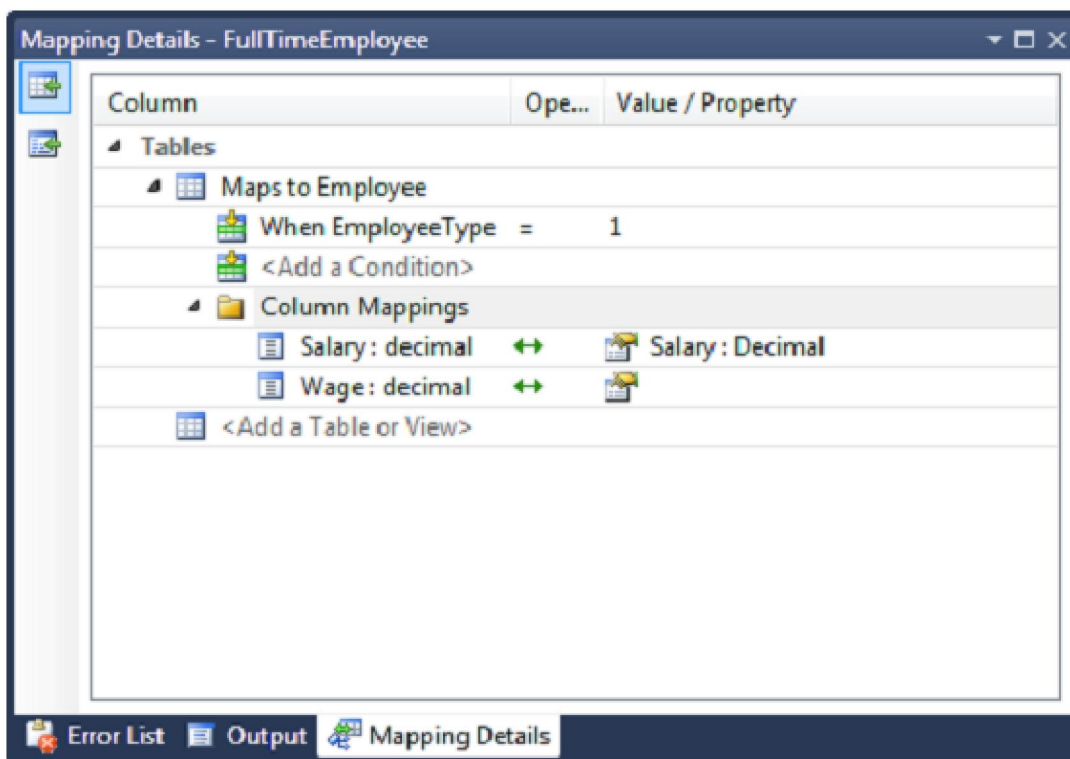


图28 FullTimeEmployee 映射详细信息

注意, 以上示例中, Employee 表中的 EmployeeType 必须为 “可以为 Null”, 否则编译可能无法通过。

最后模型如图 29 所示。

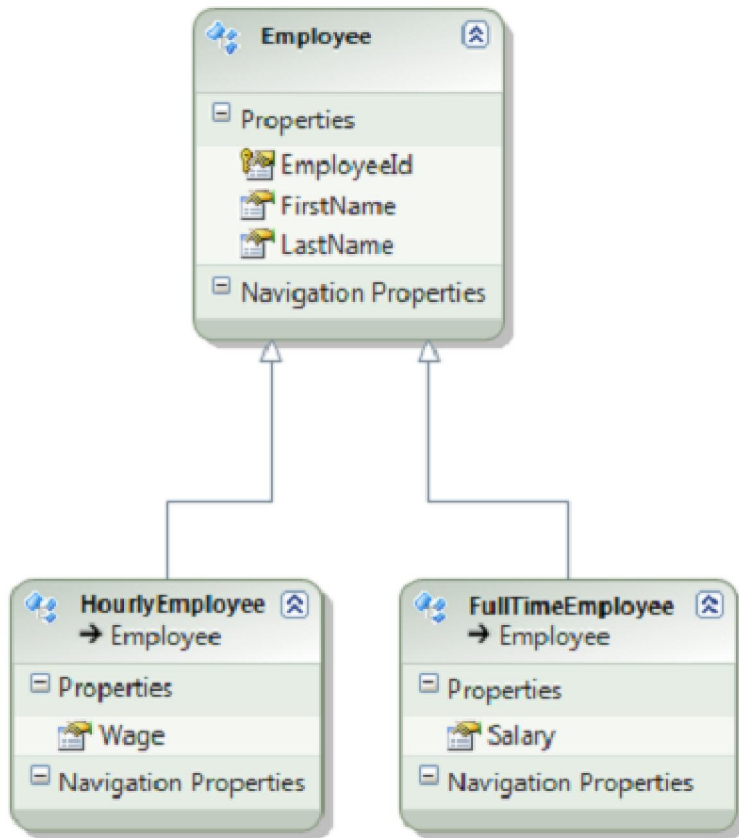


图29 最后的实体模型

代码为:

```

using (var context = new EFRecipesEntities())
{
    var fte = new FullTimeEmployee { FirstName = "Jane", LastName = "Doe",
                                     Salary = 71500M};
    context.Employees.AddObject(fte);
    fte = new FullTimeEmployee { FirstName = "John", LastName = "Smith",
                                 Salary = 62500M };
    context.Employees.AddObject(fte);
    var hourly = new HourlyEmployee { FirstName = "Tom", LastName = "Jones",
                                      Wage = 8.75M };
    context.Employees.AddObject(hourly);
    context.SaveChanges();
}

using (var context = new EFRecipesEntities())
{
    Console.WriteLine("--- All Employees ---");
    foreach (var emp in context.Employees)
    {
        bool fullTime = emp is HourlyEmployee ? false : true;
        Console.WriteLine("{0} {1} {2}", emp.FirstName, emp.LastName,
  
```

```

        fullTime ? "Full Time" : "Hourly");
    }

    Console.WriteLine("--- Full Time ---");
    foreach (var fte in context.Employees.OfType<FullTimeEmployee>())
    {
        Console.WriteLine("{0} {1}", fte.FirstName, fte.LastName);
    }

    Console.WriteLine("--- Hourly ---");
    foreach (var hourly in context.Employees.OfType<HourlyEmployee>())
    {
        Console.WriteLine("{0} {1}", hourly.FirstName, hourly.LastName);
    }
}

```

运行结果为:

```

--- All Employees ---
Jane Doe (Full Time)
John Smith (Full Time)
Tom Jones (Hourly)
--- Full Time ---
Jane Doe
John Smith
--- Hourly ---
Tom Jones

```

九、 在两个实体间创建“是”和“有”关系

在如图 30 所示的两个表间，如果同时存在“是”和“有”两种关系，可以通过以下方案实现。

在如图 30 所示的两个表中，所有可管理的“地点”放在表 Location 中，所有的“停车场”放在表 Park 中，但同时，停车场的管理办公室也必须是设置在一个“地点”，这个“地点”可以是和停车场在一起，也可以是不和停车场在一起，而且对于多个近距离的停车场，还可以共用同一个办公室。此类相似的数据关系有同时存在“是”和“有”关系，首先，停车场“是”在一个地点，这个地点的相关信息放在了表 Location 中（通过表之间的一对一关系实现），其次，停车场还“有”一个管理办公室，这个管理办公室的地点信息也放在表 Location 中。

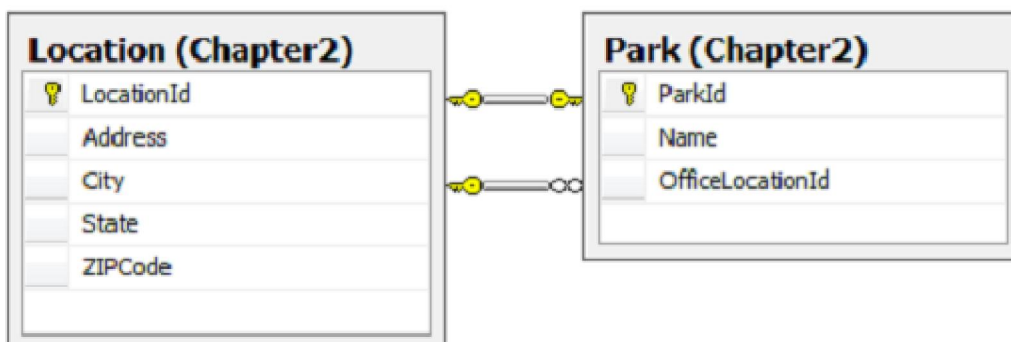


图30 表间同时存在“是”和“有”两种关系

实现操作：

1. 通过向导，向模型添加两个表。
 2. 删除向导自动添加的“1 对 1，0”关系。
 3. 右击 Location 实体，选择“添加->派生”，再选择 Park 为派生类，Location 为基类。
 4. 删除 Park 实体中的 ParkId 属性。
 5. 查看 Park 实体的映射详细信息，设置 ParkId 属性映射到 LocationId 列。
 6. 修改 Park 实体中的属性名 Location1 为 Office，用于指代停车场管理办公室。
- 实体模型修改后如图 31 所示。

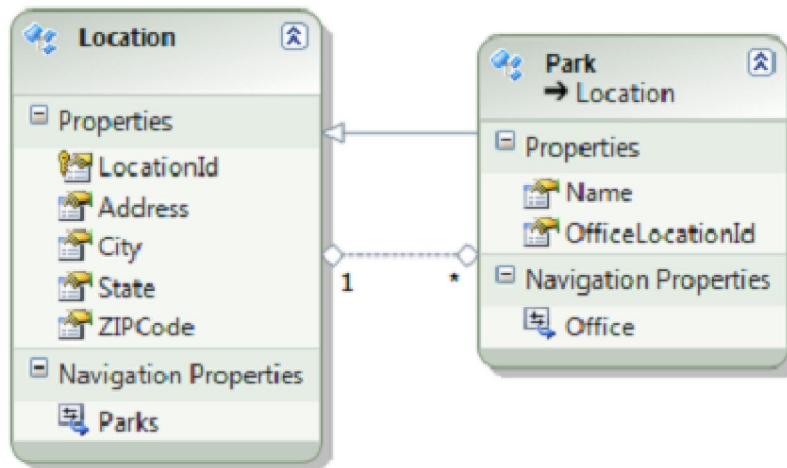


图31 同时存在“是”和“有”关系的实体模型

实现代码为：

```
using (var context = new EFRecipesEntities())
{
    var park = new Park { Name = "11th Street Park",
                        Address = "801 11th Street", City = "Aledo",
                        State = "TX", ZIPCode = "76106" };
    var loc = new Location { Address = "501 Main", City = "Weatherford",
                            State = "TX", ZIPCode = "76201" };
    park.Office = loc;
    context.Locations.AddObject(park);
    park = new Park { Name = "Overland Park", Address = "101 High Drive",
                    City = "Springtown", State = "TX", ZIPCode = "76081" };
    loc = new Location { Address = "8705 Range Lane", City = "Springtown",
                        State = "TX", ZIPCode = "76081" };
    park.Office = loc;
    context.Locations.AddObject(park);
    context.SaveChanges();
}

using (var context = new EFRecipesEntities())
{
    context.ContextOptions.LazyLoadingEnabled = true;
```

```
Console.WriteLine("-- All Locations -- ");
foreach (var l in context.Locations)
{
    Console.WriteLine("{0}, {1}, {2} {3}", l.Address, l.City,
        l.State, l.ZIPCode);
}

Console.WriteLine("--- Parks ---");
foreach (var p in context.Locations.OfType<Park>())
{
    Console.WriteLine("{0} is at {1} in {2}", p.Name, p.Address, p.City);
    Console.WriteLine("\tOffice: {0}, {1}, {2} {3}", p.Office.Address,
        p.Office.City, p.Office.State, p.Office.ZIPCode);
}
}
```

运行结果为:

```
-- All Locations --
501 Main, Weatherford, TX 76201
801 11th Street, Aledo, TX 76106
8705 Range Lane, Springtown, TX 76081
101 High Drive, Springtown, TX 76081
--- Parks ---
11th Street Park is at 801 11th Street in Aledo
    Office: 501 Main, Weatherford, TX 76201
Overland Park is at 101 High Drive in Springtown
    Office: 8705 Range Lane, Springtown, TX 76081
```