# ****一对多关系****

项目中最常用到的就是一对多关系了。Code First对一对多关系也有着很好的支持。很多情况下我们都不需要特意的去配置，Code First就能通过一些引用属性、导航属性等检测到模型之间的关系，自动为我们生成外键。观察下面的类：

[IMG_256](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_257

IMG_258

public class Destination

{

public int DestinationId { get; set; }

public string Name { get; set; }

public string Country { get; set; }

public string Description { get; set; }

public byte[] Photo { get; set; }

public List<Lodging> Lodgings { get; set; }

}

public class Lodging

{

public int LodgingId { get; set; }

public string Name { get; set; }

public string Owner { get; set; }

public bool IsResort { get; set; }

public decimal MilesFromNearestAirport { get; set; }

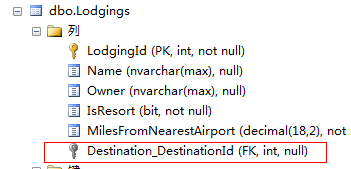
public Destination Destination { get; set; }

}

IMG_259

[IMG_260](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

Code First观察到Lodging类中有一个对Destination的引用属性，同时Destination中又有一个集合导航属性Lodgings，因此推测出Destination与Lodging的关系是一对多关系，所以在生成的数据库中为自动为Lodging表生成外键：



其实，只要在一个类中存在引用属性，即：

[IMG_262](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_263

IMG_264

public class Destination

{

public int DestinationId { get; set; }

public string Name { get; set; }

public string Country { get; set; }

public string Description { get; set; }

public byte[] Photo { get; set; }

}

public class Lodging

{

public int LodgingId { get; set; }

public string Name { get; set; }

public string Owner { get; set; }

public bool IsResort { get; set; }

public decimal MilesFromNearestAirport { get; set; }

public Destination Destination { get; set; }

}

IMG_265

[IMG_266](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

或一另一个类中存在导航属性：

[IMG_267](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_268

IMG_269

public class Destination

{

public int DestinationId { get; set; }

public string Name { get; set; }

public string Country { get; set; }

public string Description { get; set; }

public byte[] Photo { get; set; }

public List<Lodging> Lodgings { get; set; }

}

public class Lodging

{

public int LodgingId { get; set; }

public string Name { get; set; }

public string Owner { get; set; }

public bool IsResort { get; set; }

public decimal MilesFromNearestAirport { get; set; }

}

IMG_270

[IMG_271](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

Code First都能检测到它们之间一对多的关系，自动生成外键。

**指定外键**

当然我们也可以自己在类中增加一个外键。默认情况下，如果你的外键命名是规范的话，Code First会将的该属性设置为外键，不再自动创建一个外键，如：

[IMG_272](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_273

IMG_274

public class Destination

{

public int DestinationId { get; set; }

public string Name { get; set; }

public string Country { get; set; }

public string Description { get; set; }

public byte[] Photo { get; set; }

public List<Lodging> Lodgings { get; set; }

}

public class Lodging

{

public int LodgingId { get; set; }

public string Name { get; set; }

public string Owner { get; set; }

public bool IsResort { get; set; }

public decimal MilesFromNearestAirport { get; set; }

//外键

public int TargetDestinationId { get; set; }

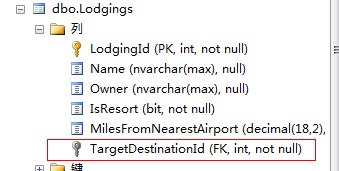
public Destination Target { get; set; }

}

IMG_275

[IMG_276](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

规范命名是指符合：命名为“[目标类型的键名]，[目标类型名称]+[目标类型键名称]”，或“[导航属性名称]+[目标类型键名称]”的形式，在这里目标类型就是Destination，相对应的命名就是：DestinationId，DestinationDestinationId，TargetDestinationId

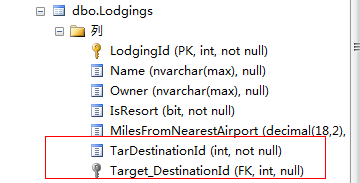


对于命名不规范的列，Code First会怎做呢？

比如我们将外键改为：

public int TarDestinationId { get; set; }

再重新生成数据库：



可以看到Code First没有识别到TarDestinationId是一个外键，于是自己创建了一个外键：Target\_DestinationId。这时我们要告诉Code First该属性是一个外键。

使用Data Annotations指定外键：

[ForeignKey("Target")]

public int TarDestinationId { get; set; }

public Destination Target { get; set; }

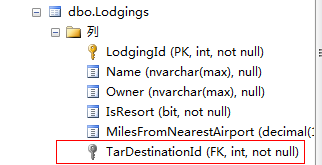
或

public int TarDestinationId { get; set; }

[ForeignKey("TarDestinationId")]

public Destination Target { get; set; }

注意ForeignKey位置的不同，其后带的参数也不同。这样，生成的数据库就是我们所期望的了。Code First没有再生成别的外键。



用Fluent API指定外键：

modelBuilder.Entity<Lodging>().HasRequired(p => p.Target).WithMany(l => l.Lodgings).HasForeignKey(p => p.TarDestinationId);

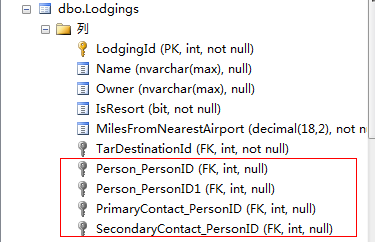
**对同一个实体多个引用的情况**

我们来考虑一下下面的情况：

IMG_280View Code

Lodging(旅店）有两个对Person表的引用，分别是PrimaryContact与SecondaryContact，同时，在Person表中也有对这两个联系人的导航：PrimaryContactFor与SecondaryContactFor。

看看Code First默认会生成怎样的数据库



天哪，竟然生成了四个外键。因为有两套类型一样的导航属性与引用属性，Code First无法确定它们之间的对应关系，就单独为每个属性都创建了一个关系。这肯定不是我们所期望的，为了让Code First知道它们之间的对应关系，在这里要用到逆导航属性来解决。

使用Data Annotations：

//第一联系人

[InverseProperty("PrimaryContactFor")]

public Person PrimaryContact { get; set; }

//第二联系人

[InverseProperty("SecondaryContactFor")]

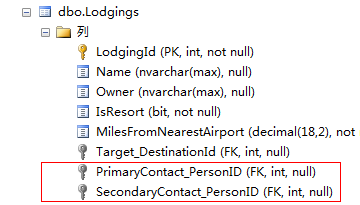
public Person SecondaryContact { get; set; }

或使用Fluent API：

modelBuilder.Entity<Lodging>().HasOptional(l => l.PrimaryContact).WithMany(p => p.PrimaryContactFor);

modelBuilder.Entity<Lodging>().HasOptional(l=>l.SecondaryContact).WithMany(p=>p.SecondaryContactFor);

再重新生成数据库，结果如图：



# ****多对多关系****

如果有两个类中，各自都是导航属性指向另一个类，Code First会认为这两个类之间是多对多关系，例如：

[IMG_283](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_284

IMG_285[酷站网软](http://www.kzwr.com/" \t "https://www.cnblogs.com/liangxiaofeng/p/_blank)

public class Activity

{

public int ActivityId { get; set; }

[Required, MaxLength(50)]

public string Name { get; set; }

public List<Trip> Trips { get; set; }

}

public class Trip

{

public int TripId{get;set;}

public DateTime StartDate{get;set;}

public DateTime EndDate { get; set; }

public decimal CostUSD { get; set; }

public byte[] RowVersion { get; set; }

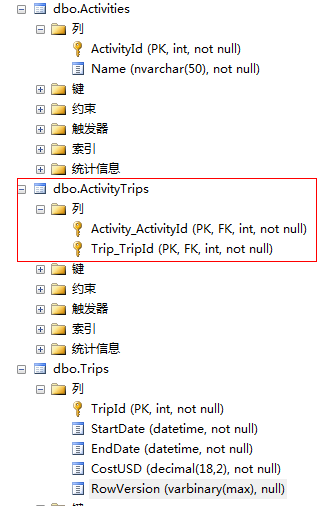
public List<Activity> Activities { get; set; }

}

IMG_286

[IMG_287](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

一个Trip类可以有一些Activites日程，而一个Activity日程又可以计划好几个trips(行程)，显然它们之间是多对多的关系。我们看看默认生成的数据库是怎么样的：



可以看到，Code First生成了一张中间表ActivityTrips，将另外两张表的主键都作为外键关联到了中间表上面。中间表中键的命名默认为"[目标类型名称]\_[目标类型键名称]".

**指定表名**

如果我们想指定中间表的名称和键名称，我们可以用Fluent API来配置。

modelBuilder.Entity<Trip>().HasMany(t => t.Activities).WithMany(a => a.Trips).Map(m =>

{

m.ToTable("TripActivities");

m.MapLeftKey("TripIdentifier");//对应Trip的主键

m.MapRightKey("ActivityId");

});

或：

[IMG_289](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

modelBuilder.Entity<Activity>().HasMany(a => a.Trips).WithMany(t => t.Activities).Map(m =>

{

m.ToTable("TripActivities");

m.MapLeftKey("ActivityId");//对应Activity的主键

m.MapRightKey("TripIdentifier");

});

[IMG_290](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

# ****一对一关系****

如果我们要将两个类配置为一对一关系，则两个类中都要配置相应的引用属性，如：

[IMG_291](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_292

IMG_293

public class Person

{

public int PersonId { get; set; }

public int SocialSecurityNumber { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

[Timestamp]

public byte[] RowVersion { get; set; }

public PersonPhoto Photo { get; set; }

}

public class PersonPhoto

{

[Key]

public int PersonId { get; set; }

public byte[] Photo { get; set; }

public string Caption { get; set; }

public Person PhotoOf { get; set; }

}

IMG_294

[IMG_295](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

我们为一个(Person)对应着一张相片(PersonPhoto),但如果根据这样的模型生成[数据库](http://www.kzwr.com/" \t "https://www.cnblogs.com/liangxiaofeng/p/_blank)为报错：

无法确定类型“BreakAway.PersonPhoto”与“BreakAway.Person”之间的关联的主体端。必须使用关系 Fluent API 或数据注释显式配置此关联的主体端

因为Code First无法确认哪个是依赖类，必须使用Fluent API或Data Annotations进行显示配置。

使用Data Annotations：

[IMG_296](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

IMG_297

IMG_298

public class Person

{

public int PersonId { get; set; }

public int SocialSecurityNumber { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

[Timestamp]

public byte[] RowVersion { get; set; }

public PersonPhoto Photo { get; set; }

}

public class PersonPhoto

{

[Key, ForeignKey("PhotoOf")]

public int PersonId { get; set; }

public byte[] Photo { get; set; }

public string Caption { get; set; }

public Person PhotoOf { get; set; }

}

IMG_299

[IMG_300](https://www.cnblogs.com/liangxiaofeng/p/javascript:void(0);)

使用Fluent API：

modelBuilder.Entity<PersonPhoto>().HasRequired(p => p.PhotoOf).WithOptional(p => p.Photo);

 注意：PersonPhoto表中的PersonId既是外键也必须是主键