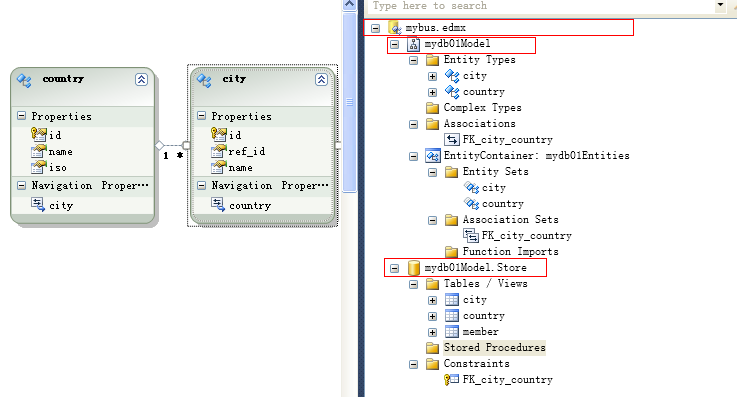
**Entity Framework**

EF 没有采取 LINQ to SQL 基于Attribute映射的做法。

为了适应变化和提供更多的数据库类型扩展,EF 提供了专门的定义语言来完成模型设置

|  |  |
| --- | --- |
| Conceptual schema definition language (.csdl) |  |
| Store schema definition language (.ssdl) |  |
| Mapping specification language (.msl) |  |

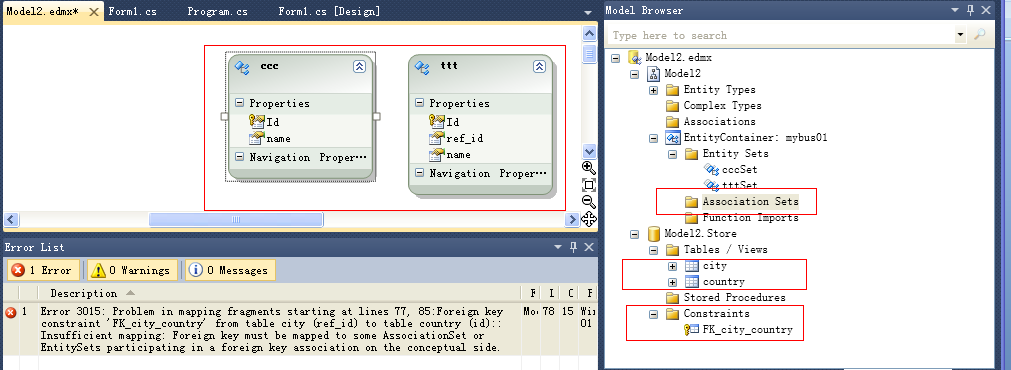
默认,Model设计器将(.csdl)(.ssdl)(.msl)存放在一个名为(.edmx)的XML 格式定义文件,并会根据设计自动生成对应的Context与实体类



**映射基本规则:**

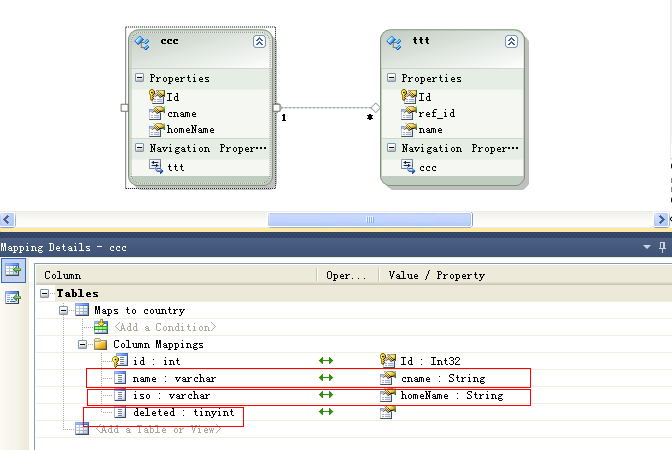
**1.[概念模型]的[数据类型],[键],[非空]约束要与[存储模型]对应**

例如： 存储模型的表之间有外键约束， 以之对应的概念模型entity之间也必须有外键约束。



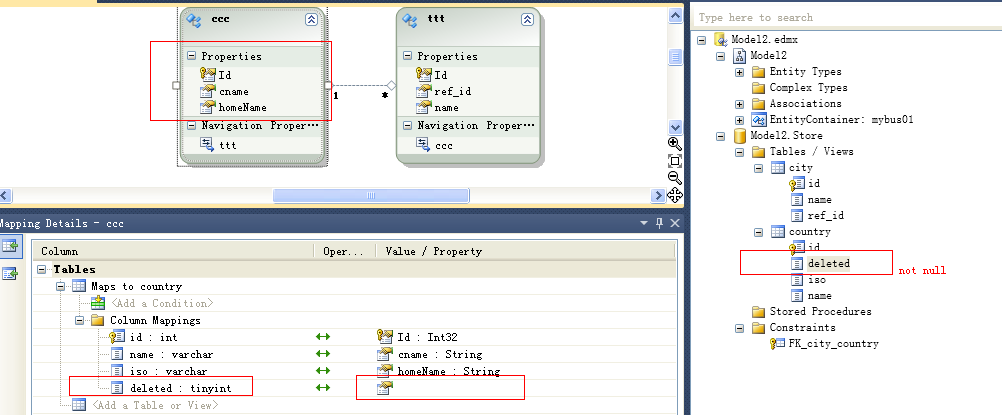
如上图： 数据库有外键， 而实体概念模型没有外键， 所以出错。

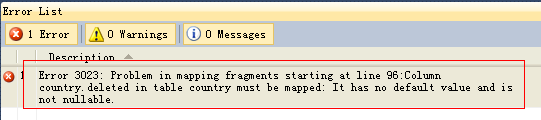
**2. [概念模型]的属性名与[存储模型]的字段名不必一致.**

****

**3.[存储模型]的字段如果有[键],[非空]约束, [概念模型]必需要有对应的属性与之映射,且类型**

**要一致。**

****

****

如上图： 在数据库 deleted 字段是非空， 如果概念模型里没有定义该字段，则出错， 必须要求在概念模型里有字段，且该字段必须也是非空，Nullable=false。

对于非唯一，唯一普通索引则没有此限制， 但是唯一索引会有问题，如果保证键值唯一， 在客户端是做不到的， 因为没有该字段存在，只能依靠服务器端的编程逻辑去实现了。

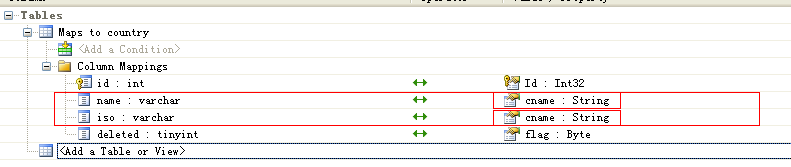
**4. [概念模型]中出现的属性,在[存储模型]中必须要有对应的字段映射，而[存储模型]可以有**

**多余的字段。如上图.**

**5. [存储模型]的多个字段可以映射到[概念模型]的同一个属性上**

　　加载时, [概念模型]的属性加载[存储模型]的第一个字段值

　　保存时, [概念模型]的属性会保存到[存储模型]的多个字段上

****

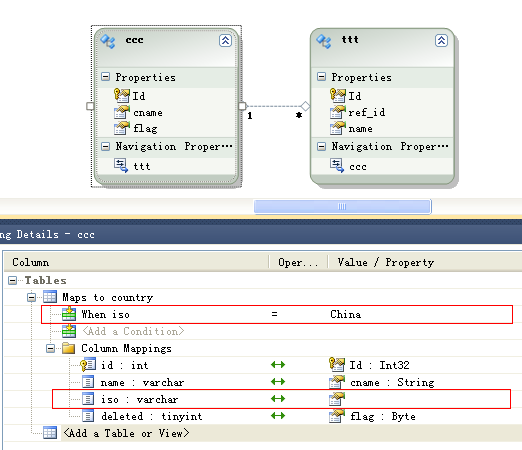
将会把 name 的值付给 cname, cname 修改以后保存会保存到数据库两个字段：name, iso

**6. 条件映射：**

　一个字段,如果要作为映射条件,就不能映射到属性上

　加载数据时,只会加载符合条件的数据

　添加数据时, 映射条件字段不可见, 映射条件字段的值是直接使用设计时的设定值来保存到数据库。

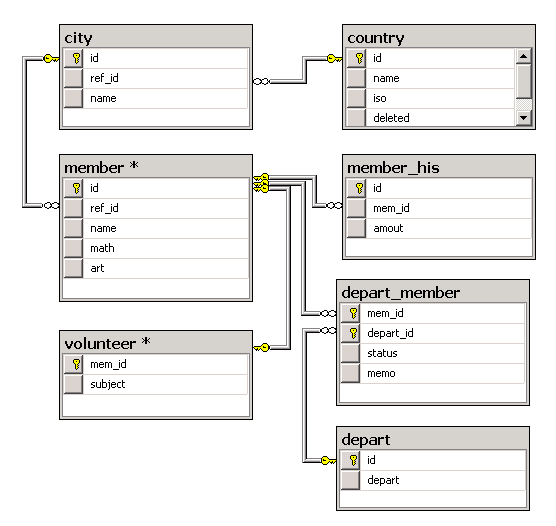
****

**6. 表类型继承：**

EF有3种方式来表达继承层次：Table per Hierarchy (TPH)，Table per Type (TPT) 和 Table per Concrete class (TPC)。

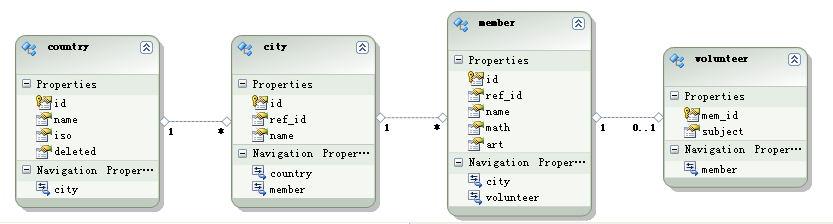
There are three different approaches to representing an inheritance hierarchy:

* **Table per Hierarchy (TPH):** Enable polymorphism by denormalizing the SQL schema, and utilize a type discriminator column that holds type information.
* [**Table per Type (TPT):**](http://weblogs.asp.net/manavi/archive/2010/12/28/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-2-table-per-type-tpt.aspx) Represent "is a" (inheritance) relationships as "has a" (foreign key) relationships.
* [**Table per Concrete class (TPC):**](http://weblogs.asp.net/manavi/archive/2011/01/03/inheritance-mapping-strategies-with-entity-framework-code-first-ctp5-part-3-table-per-concrete-type-tpc-and-choosing-strategy-guidelines.aspx) Discard polymorphism and inheritance relationships completely from the SQL schema.

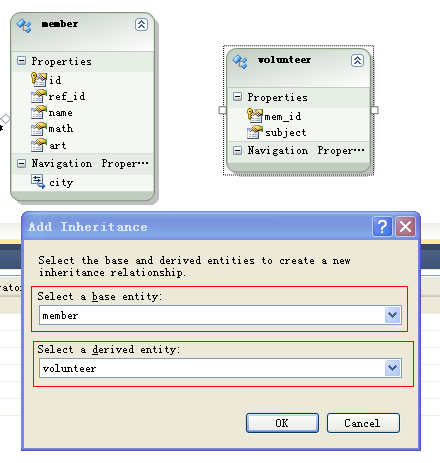
****

本例 volunteer 属于 member 的继承类。 Member( 1 ) 🡪 Volunteer( 0..1 ) 的关系。

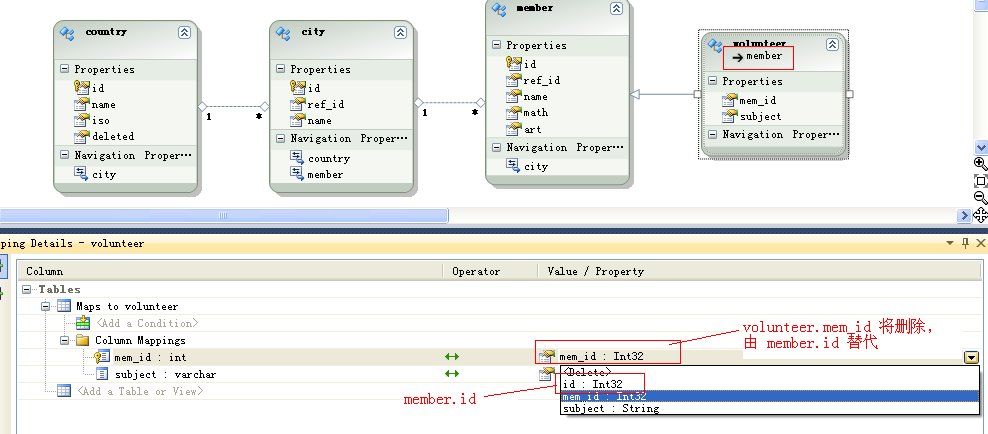
1） 将表添加进去，自动生成的Data Entity模型：



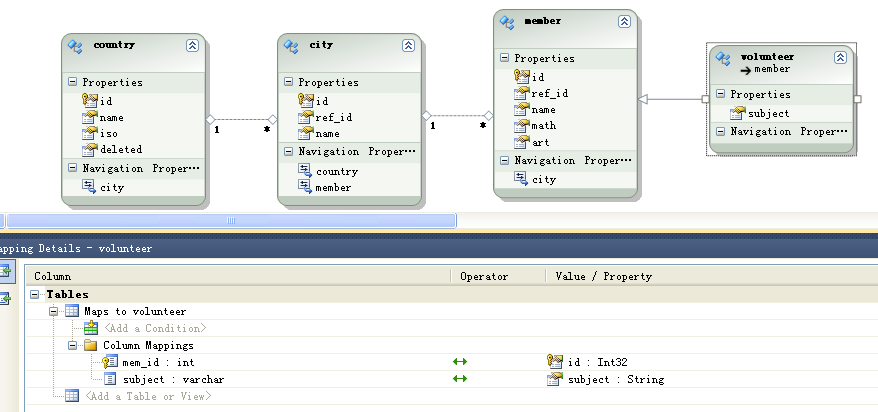
2）将自动生成的 member( 1 ) -> volunteer( 0..1 ) 的关系改成继承关系。首先删除



然后删除volunteer.mem\_id, 并且由 member.id 替代. 必须要删除，而且必须指定为member.id.



完成后如下：



关于我们是否因该将 member 设定为 abstract = true; 设定为true 对于连接其他表也没有影响。

Abstract = true; 对于表继承来说， 基表不一定要设定 abstract = true;

db1 = new mydb1();

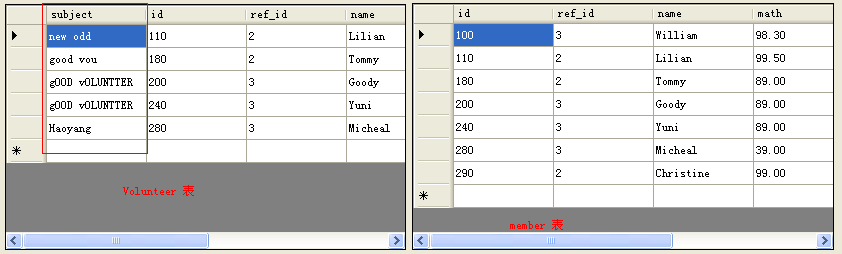
this.dgv\_vvv.DataSource = db1.member;

this.dgv\_country.DataSource = db1.member.OfType<volunteer>();

访问 member 表 ： db1.member

访问 volunteer 表： 注意没有 db1.volunteer; 只能通过 db1.member.OfType<volunteer>()

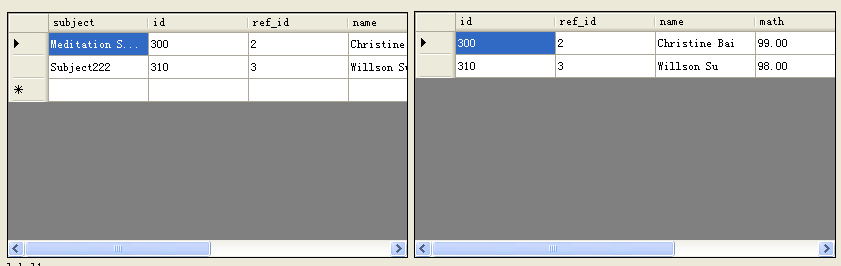
返回的是 ObjectQuery<volunteer>



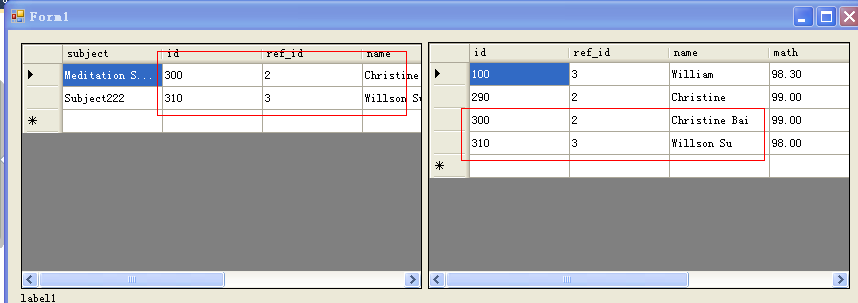
**以下情况是 abstract = true; 将基表设定为抽象类：**

我们看到以下 member 和 volunteer 的集合的记录数是一样的。 唯一不同就是 volunteer 包含

Member的信息， 而 member 没有 volunteer.subject 这个属于volunteer 的字段。



**我们看看如果 abstract = false, 基表是实体类。如下图**



**所以通常建议设定 abstract = false; 尽量保留member 的全集。除非有特殊应用**

**查询操作代码：**

member mm = db1.member.Where(p => p.name.Contains(this.textBox1.Text)).FirstOrDefault();

string str = null;

if (mm != null)

{

str += string.Format("Found Member ID: {0} Name: {1}\n", mm.id, mm.name);

}

else

{

str += string.Format("Member not found!");

}

volunteer vv = db1.member.OfType<volunteer>().Where(p =>

p.name.Contains(this.textBox1.Text)).FirstOrDefault();

if (vv != null)

{

str += string.Format("Found Volunteer ID: {0} Name: {1}\n", vv.id, vv.name);

}

else

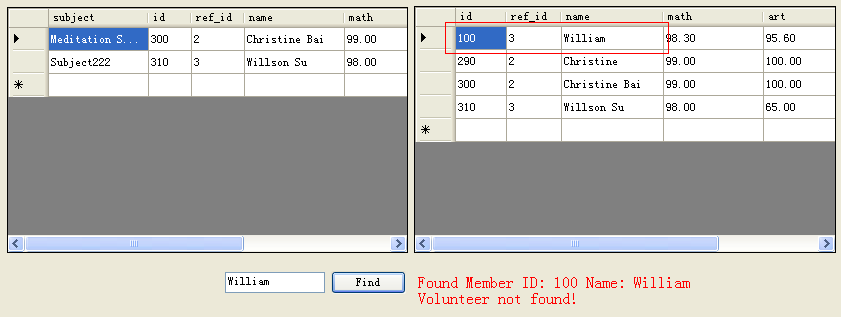
{

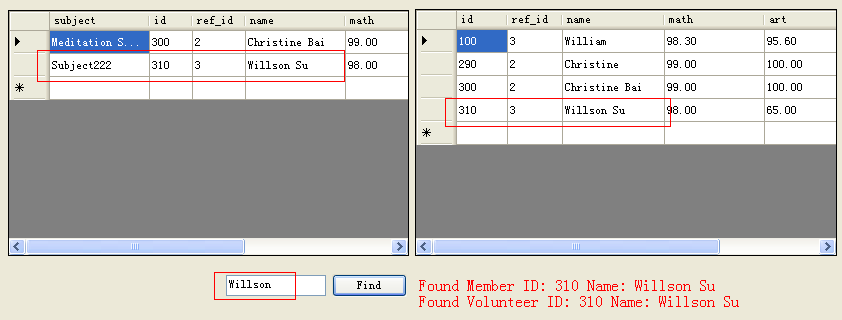
str += string.Format("Volunteer not found!");

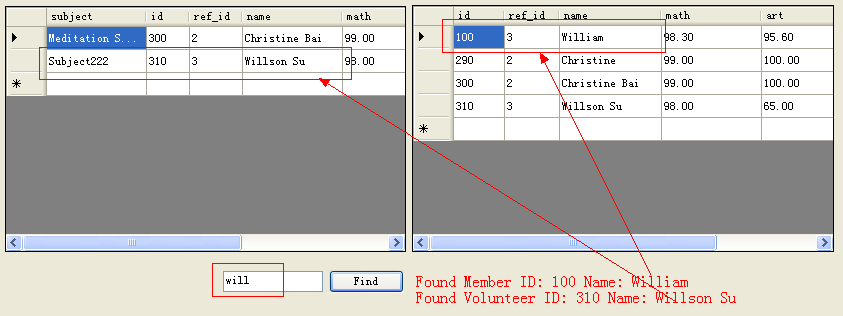
}

this.label1.Text = str;

输出结果：







**插入操作的代码：**

// member 只被插入到 member 表， volunteer 则先插入到member, 然后插入到volunteer表

member nm = new member()

{

name = "Racheal",

math = 99,

art = 100

};

city cc = db1.city.FirstOrDefault(p=>p.id == 3);

nm.city = cc;

volunteer nv = new volunteer()

{

name = "Vivian",

subject = "English Group",

math = 30,

art = 20,

city = db1.city.FirstOrDefault(p => p.id == 2)

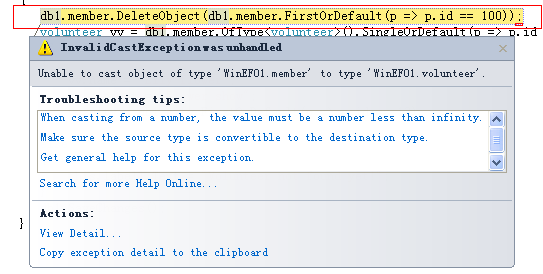
};

db1.member.AddObject(nm);

db1.member.AddObject(nv);

db1.SaveChanges();

**删除操作的代码：**

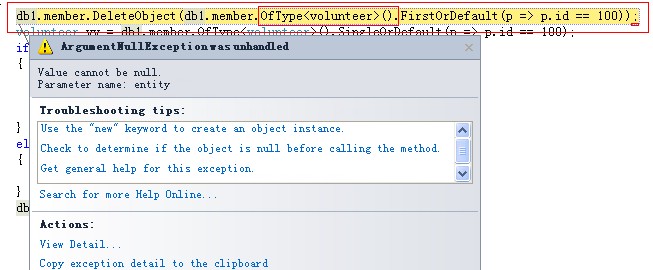


注意无论 abstract = true/false, 如果我们试图删除 member 实体则出错。

试图删除 member 一并删除 volunteer 的做法是行不通，出错的。

只能删除volunteer，并且一并删除对应的 member.

删除必须存在的实体才可以， null 也出错。



**修改操作：**

通过 member 可以只修改member 的实体，实体可以没有对应的 volunteer;

通过 volunteer 可以一并修改 member 的相关字段内容。

member mm = db1.member.FirstOrDefault(p => p.id == 100);

if (mm != null)

{

mm.math = 66;

mm.art = 88;

mm.name = "Genius";

mm.ref\_id = 20;

}

else

{

this.label1.Text = "Member not found yet!";

}

volunteer vv = db1.member.OfType<volunteer>().SingleOrDefault(p => p.id == 410);

if (vv != null)

{

vv.name = "Lily flower";

vv.subject = "Flower show";

vv.city = db1.city.FirstOrDefault(p => p.id == 13);

}

else

{

this.label1.Text = "Volunteer not found yet!";

}

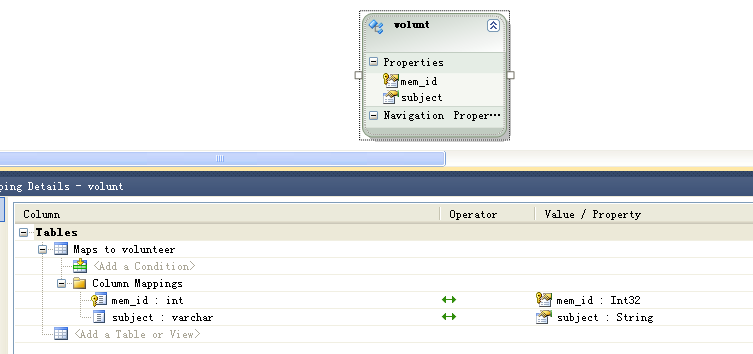
db1.SaveChanges();

**还有一种情况处理不了：**

如果 volunteer 需要新增的实体，在 member 集合里已经存在。 我们不能使用 AddObject( entity volunteer) 这样会在 member 集合里同时新增一条记录，使得记录重复出项在member 集合里：

解决方法：

1. 通过判断 确定新增的记录在 member 集合里存在， 而在 volunteer 里不存在。
2. 创建另外一个 数据模型， 专门用于操作 没有任何关系约束的 volunteer 表。组建新的记录实体， 添加即可。



**为了区分两个不同的volunteer 类：所以一个起名**volunteer**, 另个改为**volunt

**以下是完美的插入操作代码：**

member mm = db1.member.FirstOrDefault(p => p.name.Contains(this.textBox1.Text));

if (mm != null)

{

volunteer vv0 = db1.member.OfType<volunteer>().FirstOrDefault(p => p.id == mm.id);

if (vv0 == null)

{

volunt vv = new volunt()

{

mem\_id = mm.id,

subject = "New Add, But not for member"

};

db2.volunteer.AddObject(vv);

db2.SaveChanges();

}

}

else

{

volunteer vv = new volunteer()

{

name = this.textBox1.Text,

math = 100,

art = 100,

subject = "Add both VV and MM",

city = db1.city.FirstOrDefault(p => p.id == 3)

};

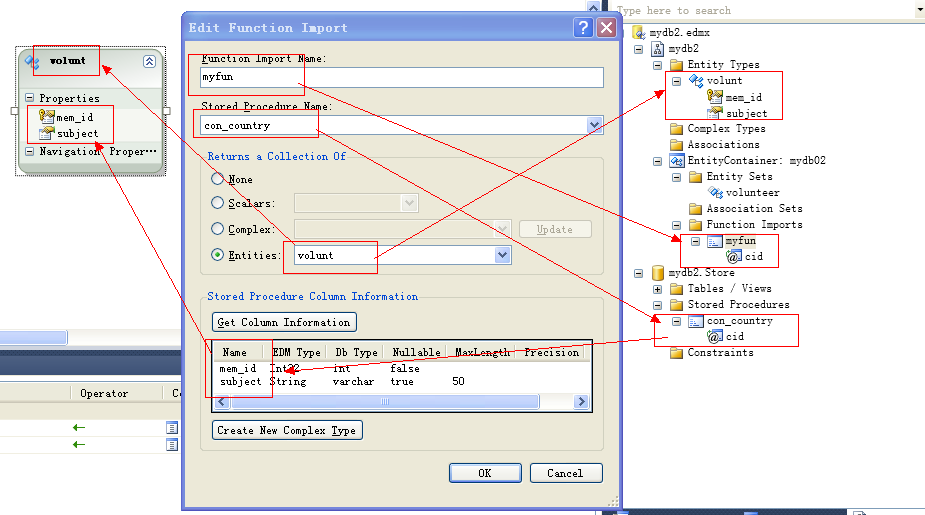
db1.member.AddObject(vv);

db1.SaveChanges();

}

**函数定义：**

**输出到实体：Entities**

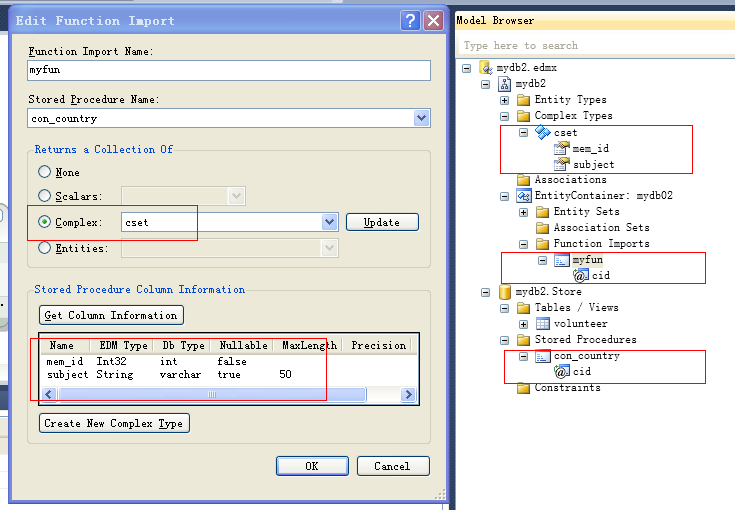


db2 = new mydb02();

ObjectResult<volunt> 　qq　 = 　db2.myfun(1);

this.dgv\_country.DataSource　 =　 qq.ToList();

输出到复合类型：



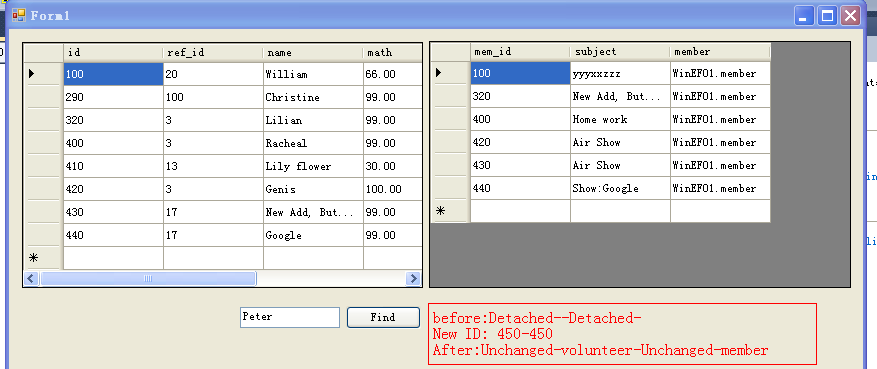
db2 = new mydb02();

ObjectResult<cset> qq = db2.myfun(1);

this.dgv\_country.DataSource = qq;

1. **[(1)—(0..1)]为主主关系,既两个表的主键关联.**

插入操作：



private void insert\_Click(object sender, EventArgs e)

{

string str = null;

member nm = new member　//如果 id 是自动号， 则不需要赋值给 id

{

ref\_id = 17,

name = this.textBox1.Text,

math = 99,

art = 96

};

volunteer nv = new volunteer // 因为是插入操作，所以 id 不需要赋值

{

subject = "Show:" + this.textBox1.Text

};

nm.volunteer = nv; // 表关系的体现

// nv.member = nm; 也可以这样实现关系

str += string.Format("before:{0}-{1}-{2}-{3}\n", nm.volunteer.EntityState, nm.volunteer.EntityKey, nm.EntityState, nm.EntityKey);

db1.member.AddObject(nm);

db1.SaveChanges();

str += string.Format("New ID: {0}-{1}\n", nm.id, nv.mem\_id);

str += string.Format("After:{0}-{1}-{2}-{3}\n", nm.volunteer.EntityState, nm.volunteer.EntityKey.EntitySetName, nm.EntityState, nm.EntityKey.EntitySetName);

this.label1.Text = str;

}

可以无关系， 各表单独插入：

主表ID 自动产生：

member nm = new member

{

ref\_id = 17,

name = this.textBox1.Text,

math = 99,

art = 96

};

// 附表 ID 必须不能违反外键约束：

volunteer nv = new volunteer

{

mem\_id = 400,

subject = "Show:" + this.textBox1.Text

};

// 附表 ID 必须不能违反外键约束：

volunteer nv1 = new volunteer()

{

mem\_id = 410,

subject = "show1:" + this.textBox1.Text

};

插入时重复对附表赋值， 只取最后赋值的插入：

volunteer nv = new volunteer

{

subject = "Show:" + this.textBox1.Text

};

volunteer nv1 = new volunteer()

{

subject = "show1:" + this.textBox1.Text

};

nm.volunteer = nv;

nm.volunteer = nv1; //只有这条记录插入， 覆盖上一个值。

**删除操作：**

private void delete\_Click(object sender, EventArgs e)

{

string str = null;

member mm = db1.member.FirstOrDefault(p => p.name.Contains(this.textBox1.Text));

if (mm != null) {

str += string.Format("Member Found: {0} - {1}\n", mm.id, mm.name);

db1.member.DeleteObject(mm);

} else { str += string.Format("Member not found!\nVolunteer not found!"); }

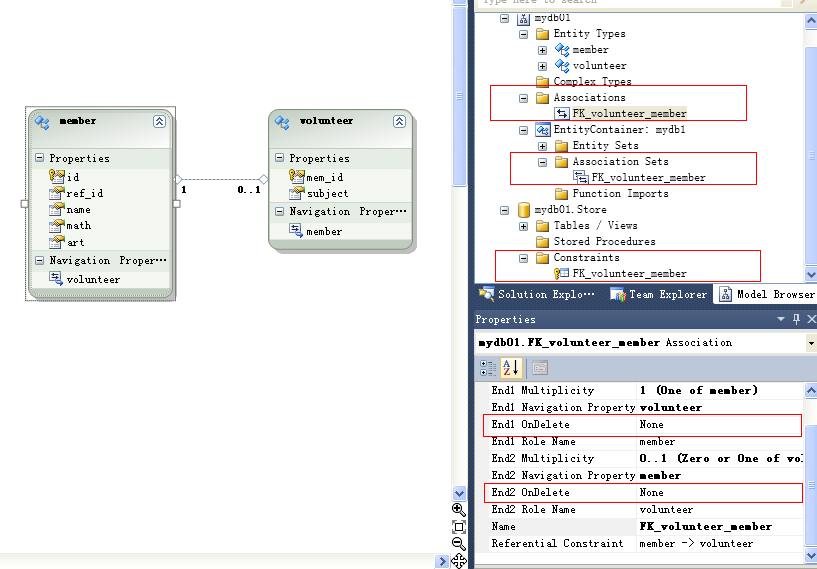
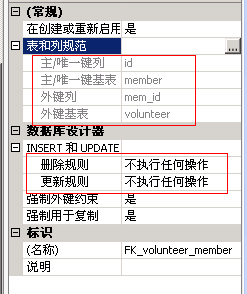
db1.SaveChanges();

this.dgv\_vvv.DataSource = db1.member;

this.dgv\_country.DataSource = db1.volunteer;

}

**如下图： 虽然是实体定义和数据库定义里都对删除操作的规则设定都是不处理，但是对主表删除同时删除附表的记录。 Delete 规则不起作用。**

**** 

直接删除不带附表记录的主表记录，

只删除附表的记录：只需要将附表记录赋值为 null , 保存即可。

string str = null;

member mm = db1.member.FirstOrDefault(p => p.name.Contains(this.textBox1.Text));

if (mm != null) {

str += string.Format("Member Found: {0} - {1}\n", mm.id, mm.name);

if (mm.volunteer != null) {

mm.volunteer = null; // 此处只删除附表记录， 直接赋值为 null

} else {

str += string.Format("Volunteer not found!");

}

} else {

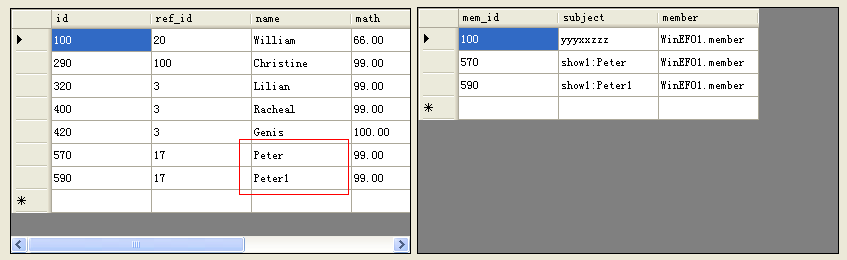
str += string.Format("Member not found!\nVolunteer not found!");

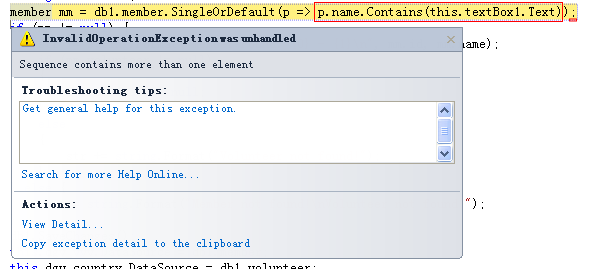
}

db1.SaveChanges();

SingleOrDefault 与 FirstOrDefault 的区别：

1. 两者都是返回实体集合里的单个实体。
2. SingleOrDefault 要求查询过滤条件必须返回是单个实体，如果返回的是多个实体，则报错：





1. FirstOrDefault 查询条件返回空，单个或者多个实体都可以。但是只头一位实体返回。

member mm = db1.member.FirstOrDefault(p => p.name.Contains(this.textBox1.Text));

所以使用FirstOrDefault 就不会有上面的错误发生。

**修改操作：**

private void Edit\_Click(object sender, EventArgs e)

{

cnt++;

string str = null;

member mm = db1.member.SingleOrDefault(p => p.id == this.fid);

if (mm != null) {

mm.name = "changed:" + cnt.ToString() + ":Lilian"; // 修改主表内容

volunteer vv = new volunteer() { subject = "Sub:" + cnt.ToString() + ":" + this.textBox1.Text };

volunteer vv1= new volunteer() { subject = "VVV:" + cnt.ToString() + ":" + this.textBox1.Text };

mm.volunteer = vv;

mm.volunteer = vv1; /**/ 修改附表内容，不管原来附表存在不存在记录，不管赋值多少次，以最后一次为主。**

**有则改之，无则加入。**

} else {

str += string.Format("Member not found!\nVolunteer not found!");

}

db1.SaveChanges();

this.label1.Text = str;

}

**附表如果给关系的主键赋值： 将出错**

volunteer vv = new volunteer()

{

mem\_id=400, // 将出错， 除非该值刚好等于其主表的关键字的值。此时不违反约束。则不出错。

// 通常是不需要给 id 赋值的。

subject = "Sub:" + cnt.ToString() + ":" + this.textBox1.Text

};

mm.volunteer = vv;

**另外一种处理办法， 附表有记录则修改现有值得， 没有则添加。**

member mm = db1.member.SingleOrDefault(p => p.id == this.fid);

if (mm != null)

{

mm.name = "change:" + cnt.ToString() + this.textBox1.Text;

if (mm.volunteer != null) {

// mm.volunteer.mem\_id = 400; // 不能更改关系的ID 值。出错。

mm.volunteer.subject = "Sub:" + cnt.ToString() +":" + this.textBox1.Text; // 更改现值。

} else {

volunteer vv = new volunteer {

subject = "New:" + cnt.ToString() + ":" + this.textBox1.Text

};

mm.volunteer = vv; // 添加

}

} else {

str += string.Format("Member not found!\nVolunteer not found!");

}

db1.SaveChanges();

**2. [(1)—(1)]为主主关系,既两个表的主键关联**

跟上面 (1) - (0..1) 操作上是一样的。

但是理论上： 两者是主主的关系。 所以？？？

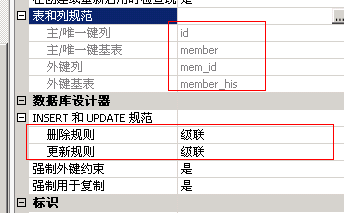
1. **[(1)—(\*)] 关联：**

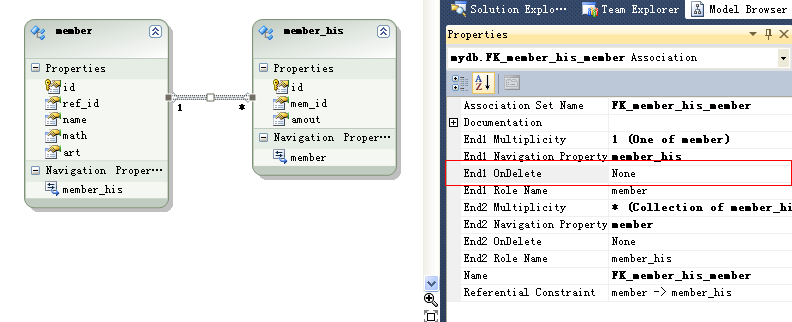
更改操作：

1)主键不允许修改

删除操作：

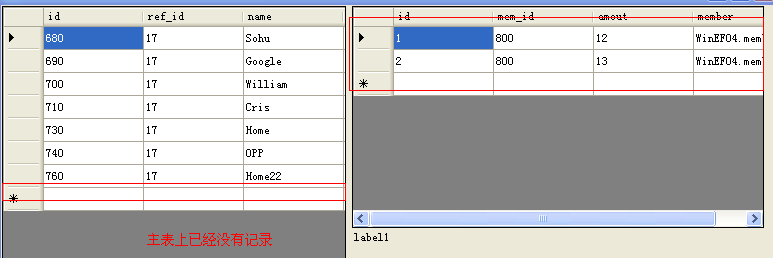
1. 如果主表和子表之间的删除规则是 None, 这不管数据库有没有此规则， 在客户端级别就会出错



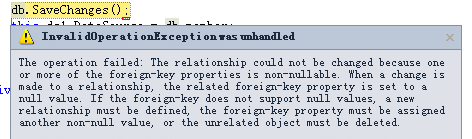


删除的时候不出错，主表记录状态被改， 而子表则不受影响。

db.member.DeleteObject( db.member.SingleOrDefault(p => p.id == 800) );

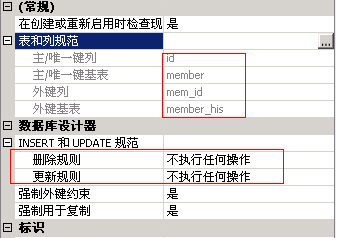


但是我们执行保存时候，出错了

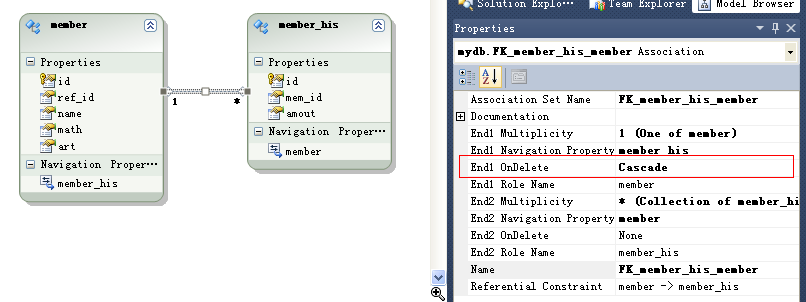


2）主表和子表的删除规则设定为 Cascade, 则不管数据库有没有此规则，即使没有， 也同样可

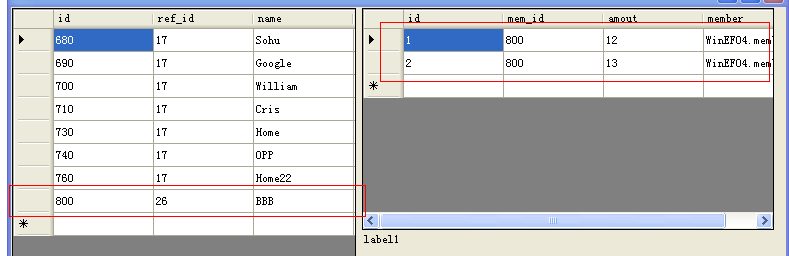
以删除子表的级联记录。 但是在数据库端，主表违反了和子表以外其他表的约束，也会出错。



生成数据模型时，默认跟数据库是一样的， 现在手工改成 Cascade .



删除以前：



删除以后：



删除子表：

int id = -1;

int.TryParse(this.textBox1.Text, out id);

member mm = db.member.SingleOrDefault(p => p.id == id);

member\_his mh = mm.member\_his.FirstOrDefault(p => p.amout >= 65);

if (mh != null)

db.member\_his.DeleteObject(mh);

else

MessageBox.Show("Delete Complete!");

db.SaveChanges();

this.dg1.DataSource = db.member.ToList();

this.dg2.DataSource = db.member\_his.ToList();

删除只能在 Container 的相关表里删除：

db.member\_his.DeleteObject(mh);

注意以下方法都不能使用：

1.

mm.member\_his.FirstOrDefault(p => p.amout >= 65) = null;

2.

member\_his mh = mm.member\_his.FirstOrDefault(p => p.amout >= 65);

mm.member\_his.Remove(mh);

修改则较简单：

int id = -1;

int.TryParse(this.textBox1.Text, out id);

member mm = db.member.SingleOrDefault(p => p.id == id);

mm.name = "change:" + mm.name;

member\_his mh = mm.member\_his.FirstOrDefault(p => p.amout >= 65);

mh.amout += 120;

//if (mh != null) db.member\_his.DeleteObject(mh); else MessageBox.Show("Delete Complete!");

db.SaveChanges();

注意以下的问题：

this.label1.Text = mm.member\_his.Count().ToString() + ":" + mhl.Count() + ":" +

mhl.FirstOrDefault().GetType().ToString();

IEnumerable<member\_his> mhl = mm.member\_his.DefaultIfEmpty();

如果没有子记录： mm.member\_his.Count() 返回 0；

mhl.Count() 返回 1；

mhl.FirstOrDefault().GetType()； 则出错，因为 object = null; null.getType()

添加子表：

int id = -1;

int.TryParse(this.textBox1.Text, out id);

member mm = db.member.SingleOrDefault(p => p.id == id);

if (mm != null)

{

member\_his mh = new member\_his()

{

amout = 90

};

member\_his mh1 = new member\_his()

{

amout = 75

};

member\_his mh2 = new member\_his()

{

amout = 65

};

mh.member = mm;

mm.member\_his.Add(mh1);

mm.member\_his.Add(mh2);

db.SaveChanges();

}

else

{

MessageBox.Show("Member not found");

}

this.dg1.DataSource = db.member.ToList();

this.dg2.DataSource = db.member\_his.ToList();

添加主表和子表:

member mm = new member()

{

ref\_id = 20,

name = "Tommy Su",

math = 100,

art = 98

};

member\_his mh = new member\_his()

{

amout = 90

};

member\_his mh1 = new member\_his()

{

amout = 75

};

member\_his mh2 = new member\_his()

{

amout = 65

};

mh.member = mm;

mm.member\_his.Add(mh1);

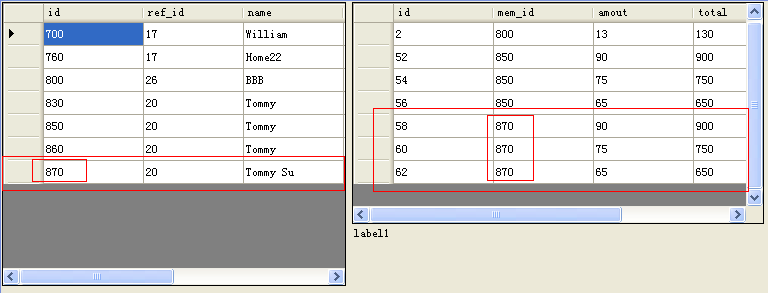
mm.member\_his.Add(mh2);

db.member.AddObject(mm);

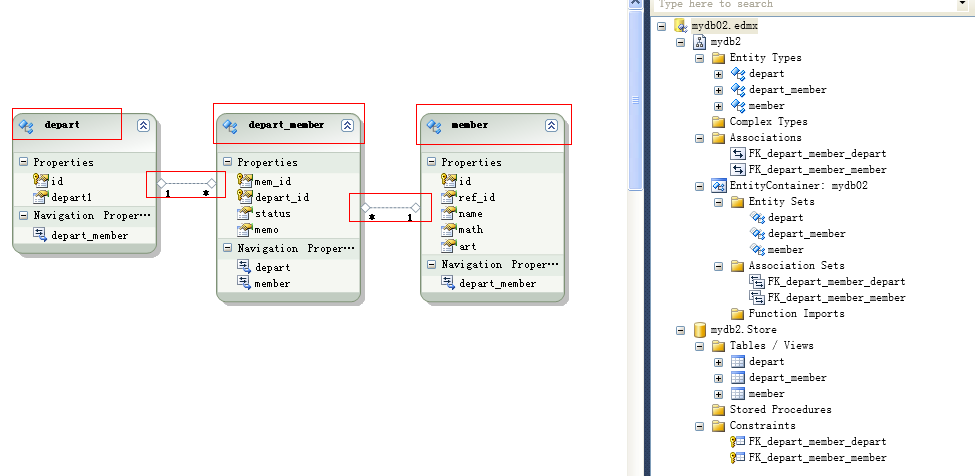
db.SaveChanges();

this.dg1.DataSource = db.member.ToList();

this.dg2.DataSource = db.member\_his.ToList();



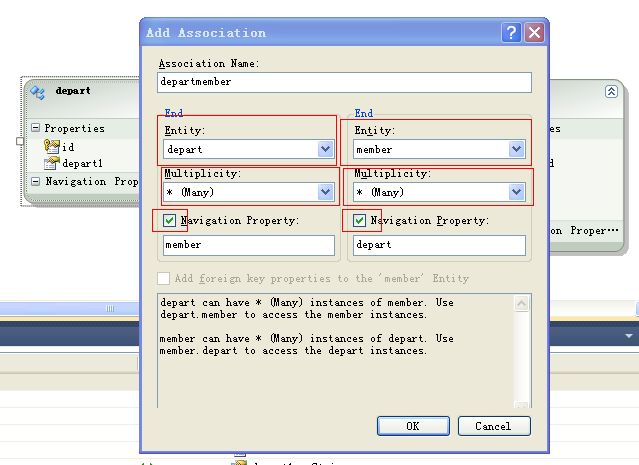
1. **[(\*)—(\*)]关联：**

****

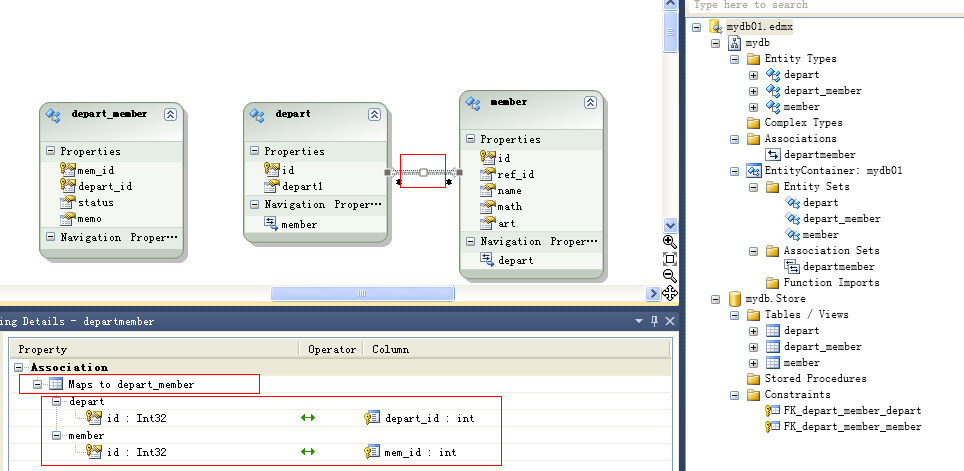
**将自动生成的关系删除掉：**

****

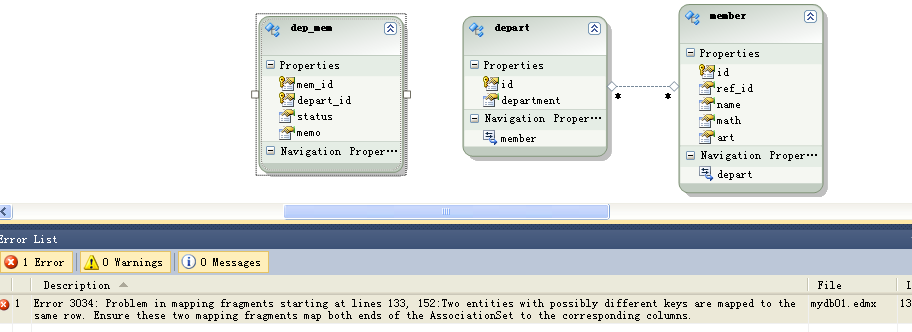
**从 depart 到 meber 添加多对多关系：**

****

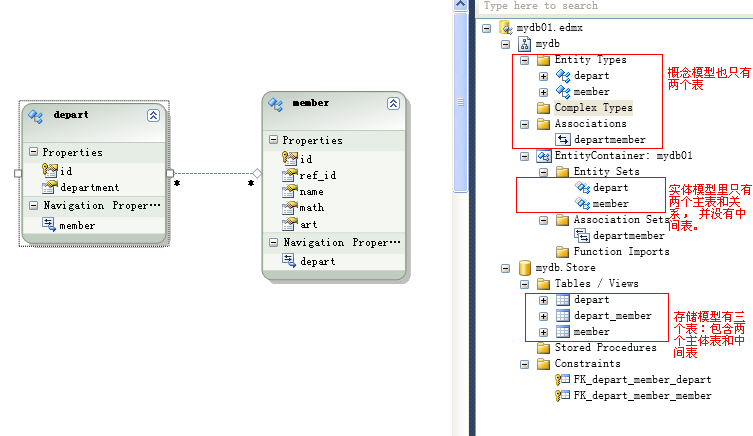
添加关系影射：需要影射到中间表：depart <-> depart\_member <-> member



必须将中间表 depart\_member 删除， 因为关系已经影射到数据库，此时中间表也重复影射到同一数据库表。 则出错：



删除后，最后的模型如下：



**多对多插入操作：**

member nm = new member() {

ref\_id = 20,

name = this.tdep.Text,

math = 9,

art = 8

};

depart nd = new depart() { department = "Depart for" + this.tdep.Text };

// 新 member 添加新 depart.

nm.depart.Add(nd);

depart dp = null;

dp = db.depart.SingleOrDefault(p => p.id == 10);

if (dp != null) // 给 depart.id = 10 添加新member;

dp.member.Add(nm);

dp = db.depart.SingleOrDefault(p => p.id == 30);

if (dp != null) // 给新member 添加 depart.id = 30;

nm.depart.Add(dp);

dp = db.depart.SingleOrDefault(p => p.id == 5);

if (dp != null) // 不添加，因为不存在。

nm.depart.Add(dp);

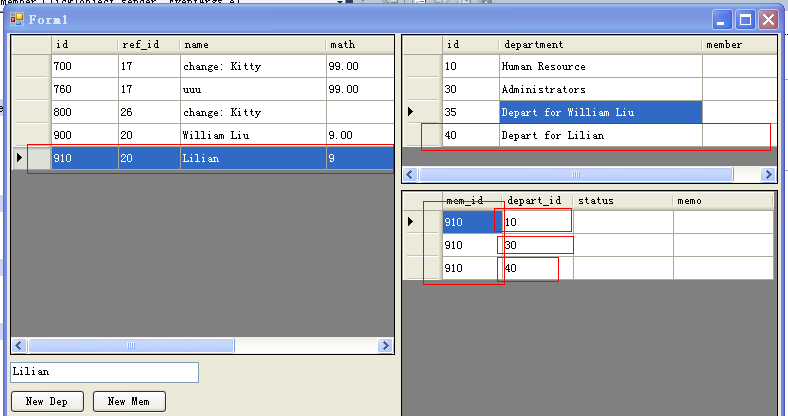
db.SaveChanges();

this.dg1.DataSource = db.depart.ToList();

this.dg2.DataSource = db.member.ToList();

this.dg3.DataSource = db2.depart\_member.ToList();

添加结果如下：



另外一种添加： 重叠双方添加， 也不出错， 记录只有一条加入到中间表。

dp = db.depart.SingleOrDefault(p => p.id == 10);

if ( dp != null ) {

dp.member.Add(nm); // 部门加人

nm.depart.Add(dp); // 人加部门

}

**重复添加： 如果对应的附表记录已经存在，注意如果是从附表里选择记录，重复添加不会出错，只是不加入而已。**

member mm = db.member.FirstOrDefault(p => p.name.Contains(this.tdep.Text));

if (mm != null)

{

MessageBox.Show(string.Format("Member Exists: has {0} departs", mm.depart.Count));

mm.depart.Add(db.depart.SingleOrDefault(p => p.id == 10));

mm.depart.Add(db.depart.SingleOrDefault(p => p.id == 30));

}

this.dbsave();

以上代码：如果记录存在则忽略不添加， 如果不存在则添加。

**把自己的附表记录，添加给其他记录的附表里。 这时是Copy , 不是移动。**

member mm = db.member.FirstOrDefault(p => p.name.Contains(this.tdep.Text)); // 记录 member: 970

if (mm != null)

{

depart dp = mm.depart.FirstOrDefault(); // 记录 depart: 10

if (dp != null) {

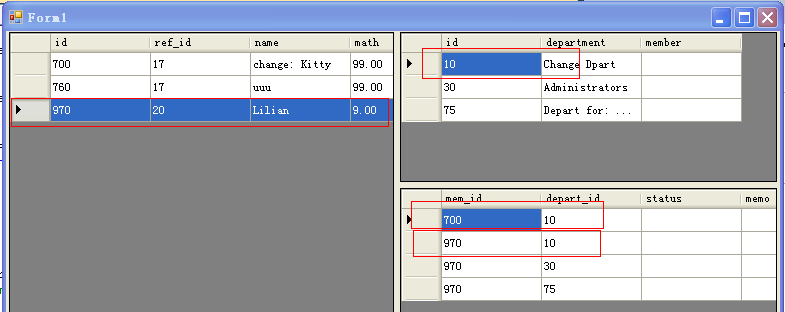
db.member.First().depart.Add(mm.depart.FirstOrDefault()); // 记录 member: 700 添加 depart: 10

}

}

this.dbsave();

结果如下：

****

**以下重复添加会出错： 如果是从主表的附表里选择出记录， 添加到其他记录的的附表里， 记录已经存在的话，会违反约束出错。 总结就是 附表记录 重复添加到 附表记录**

db.member.First().depart.Add( mm.depart.FirstOrDefault() );

member\_one.depart.Add ( member\_two.depart.FirstOrDefault() ); 相当于

**移除附表记录：**

member mm = db.member.FirstOrDefault(p => p.name.Contains(this.tdep.Text)); // 记录 member: 970

depart dp = mm.depart.FirstOrDefault();

if( dp != null ) mm.depart.Remove(); //然后移除。

db.SaveChanges();

**移动附表记录：**

member mm = db.member.FirstOrDefault(p => p.name.Contains(this.tdep.Text)); // 记录 member: 970

if (mm != null)

{

depart dp = mm.depart.FirstOrDefault(); // 记录 depart: 10

if (dp != null) {

db.member.First().depart.Add(dp); // 记录 member: 700 添加 depart: 10

mm.depart.Remove(dp); //然后移除。

}

}

this.dbsave();

**修改记录：**

member mm = db.member.FirstOrDefault(p => p.name.Contains(this.tdep.Text));

if (mm != null)

{

mm.name = "Change Name"; // 更改member.name

depart dp = mm.depart.FirstOrDefault();

if (dp != null) dp.department = "Change Dpart"; //更改member所属第一个部门的名称。

}

this.label1.Text = str;

db.SaveChanges();

**删除记录：**

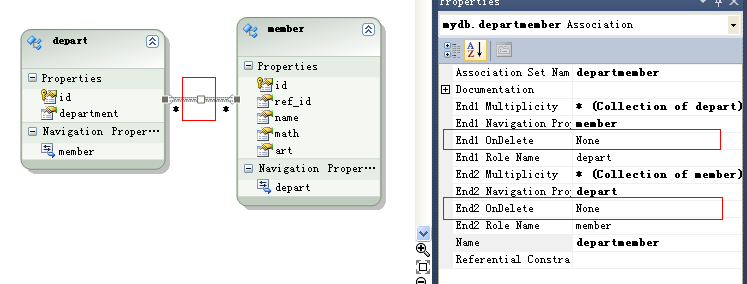
删除附表记录参考上面，

删除主表则需要注意： 多对多， 对于删除规则 多方都不能是用 Cascade. 因此如果设置Cascade 会出错。

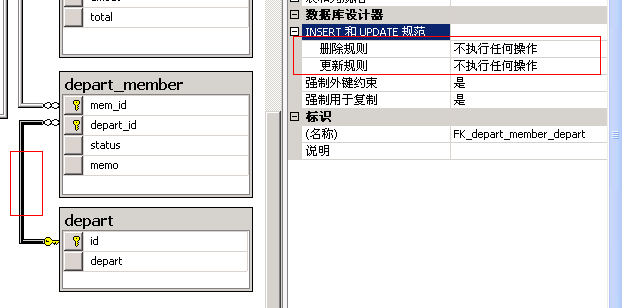
所以删除规则完全取决于 数据库端的删除规则如何设置， 如果是“不做任何操作“， 我们提交删除时会违反外键约束而出错。

我们将数据库的删除规则改为Cascade, 则删除没有任何问题。

实体模型的删除规则：



数据的删除规则



执行删除操作：

depart dp = db.depart.FirstOrDefault(p => p.department.Contains(this.tdep.Text));

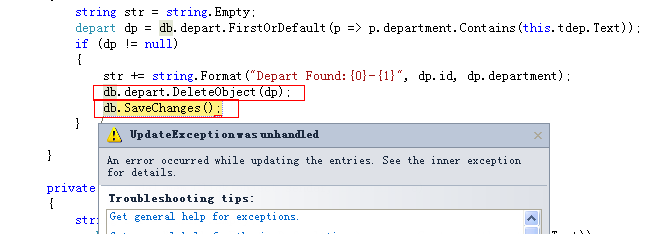
if (dp != null)

{

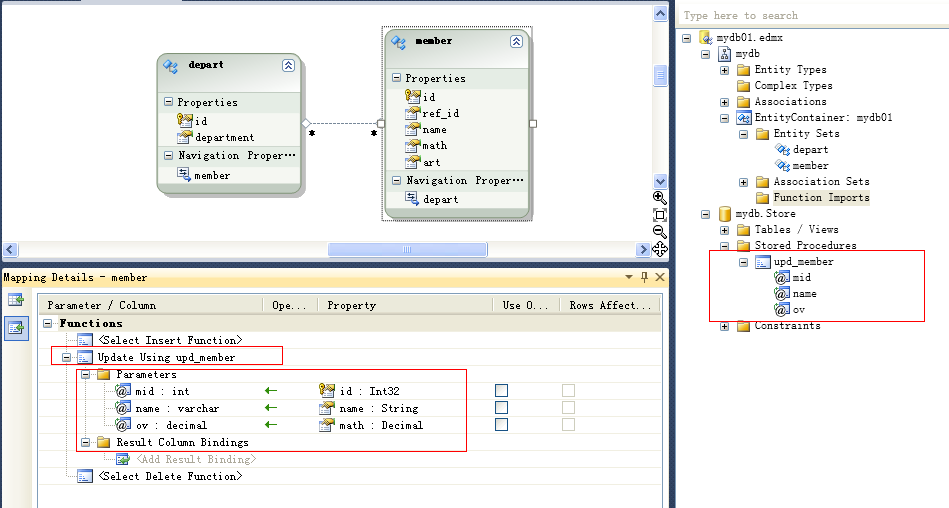
db.depart.DeleteObject(dp);

db.SaveChanges();

}



**放弃默认的增，删，改， 使用自定义存储过程：**

****

**主要是做好映射关系:**

**注意:**

**1) 如果使用自定义的插入存储过程. 则主键是自动识别号, 是由后端数据库生成的.**

**需要解决方案将此自动生成的值 同步回客户端.**

**否则如果插入操作提交以后, 客户端要使用该 ID 的值, 将出问题.**

**ObjectContext : 提供了直接执行数据库存储过程和函数的方法：**

ExecuteStoreQuery<TEntity > - ObjectResult<TEntity>

ExecuteStoreCommand( comText, parameters[] ) - int

ExecuteFunction ( funName, parameters[] ) – int

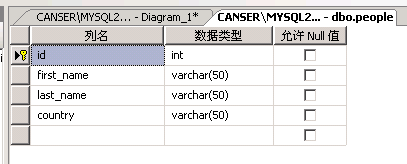
ExecuteFunction <TElement>( funName, parameters[] ) – ObjectResult<TElement>

myb.ExecuteStoreCommand("INSERT people(first\_name, last\_name, country) values('New FN','New LN', 'Canada')");

**直接操作数据库， 不需要提交。**

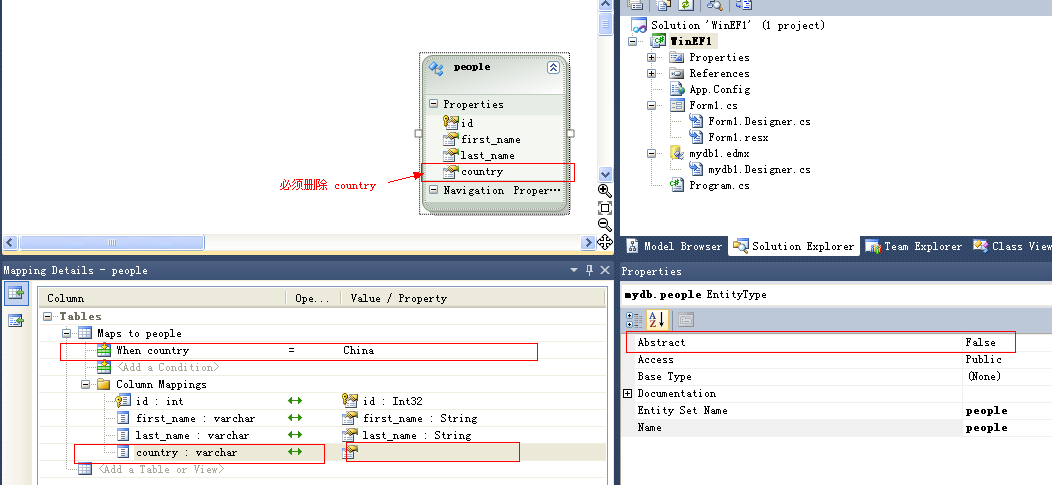
* **条件映射与继承的结合使用：**

**情况一： 表格自己使用条件**

****

**Entity Model:**

1. 条件使用的是存储模型里的字段：country
2. 概念模型不需要再定义 country, 将来回写数据库，将使用条件设定的值。
3. 在映射里也不需要将存储模型字段 country 映射到概念模型的country (而且它也不存在)。
4. 概念模型 Abstract 不能是抽象类， 必须是实体类。

****

people person = new people

{

id = 300,

first\_name = "William",

last\_name = "Liu" + DateTime.Now.ToLongTimeString()

};

myb.people.AddObject(person);

myb.SaveChanges();

MessageBox.Show("Insert id:" + person.id.ToString());

1. 此时实体的集合是一个满足条件的子集合， 所有的操作都在子集合的范围内。

// First 是指满足条件 country = China 的第一个记录。

people op = this.myb.people.FirstOrDefault();

if (op != null)

this.myb.people.DeleteObject(op);

else

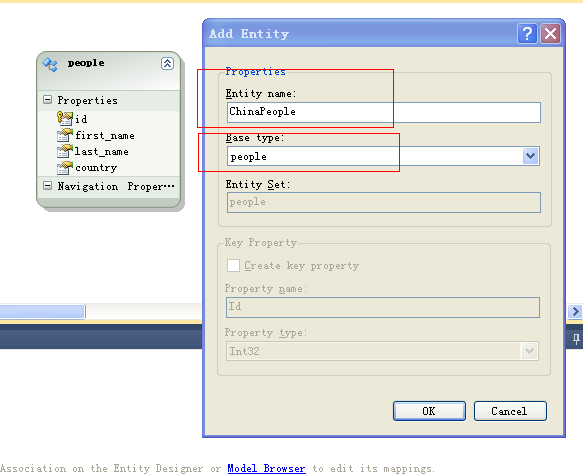
MessageBox.Show("Record can not be found");

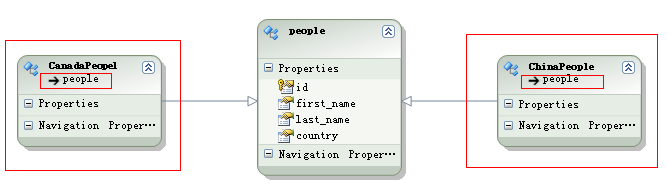
this.myb.SaveChanges();

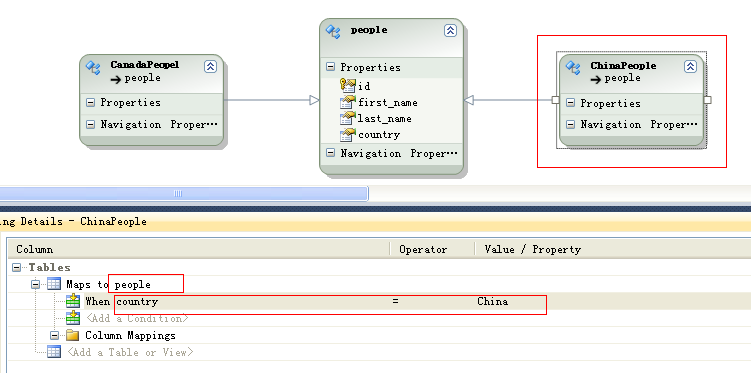
**情况二： 单个表拆分成多个不同条件的表：**

例如： people 表根据country 的不同分成： ChinaPeople, CanadaPeople … 等不同的表：

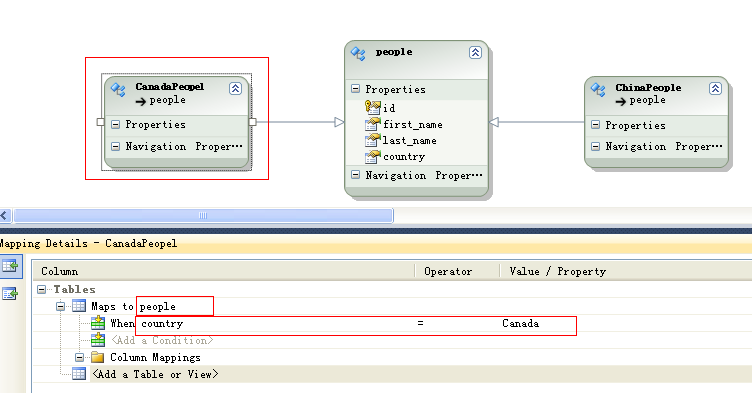
1. **添加新的Entity , 起名字叫： ChinaPeople , Base Type: people**



1. **同样加另外一个表：CanadaPeople** 
2. **对新加的Entity 做条件映射：**

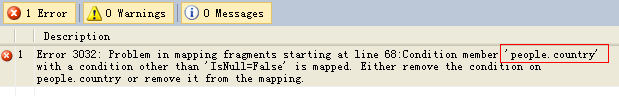


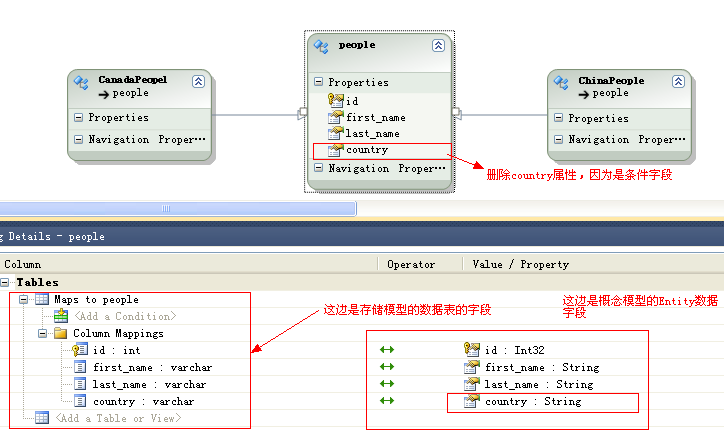
1. **对另一个表也做条件设置：**



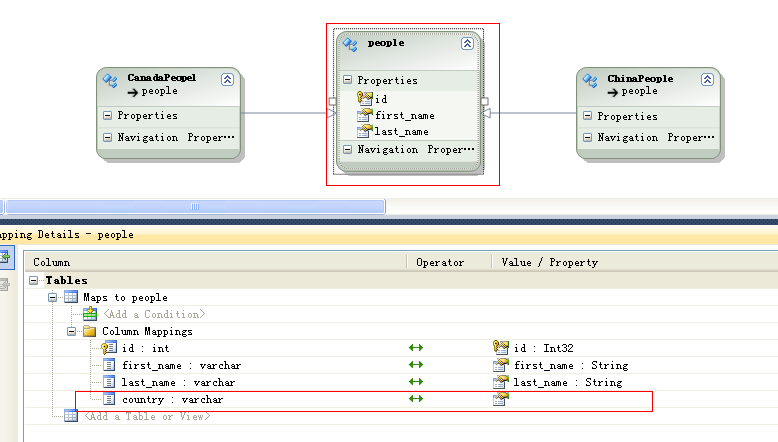
1. **Base Type: people 必须做修改:**
2. country 字段已经被作为条件使用， 所以不能出现在 people 里。

如果不删除则提示出错：



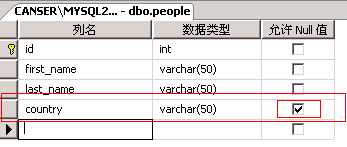


删除以后：



1. 此时 people 是实体类而不是抽象类的情况： 我们在客户端使用时， 可以直接使用 people.ToList();

如果想要 people 成为实体类： 存储模型的people表的条件字段country 必须是 nullable 允许为空：



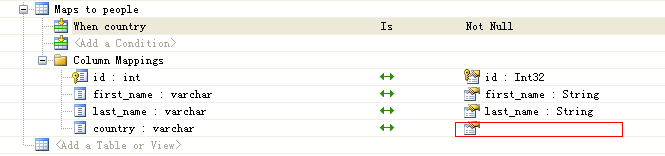
为什么people 如果是实体类（不是抽象类）,存储模型的条件字段country 必须允许空值？？？

我们回顾前面映射基本原则：第三条：country 有非空约束, 概念模型必须要与之映射. 但是此字段作为条件字段, 是不能在概念模型存在的.

**3.[存储模型]的字段如果有[键],[非空]约束, [概念模型]必需要有对应的属性与之映射,且类型要一致。**

所以必须在概念模型设置映射， 但是概念模型又不允许有该字段。

另外一个解决方案也不行， 将 country 放入条件 country is Not Null. 以避开属性映射。 这样也不行。



错误：

**所以总结：**

基类如果是实体类，不是抽象类， 必须保证条件字段在存储模型里是允许空值 Nullable .

否则出现错误。

**快速解决方案：**

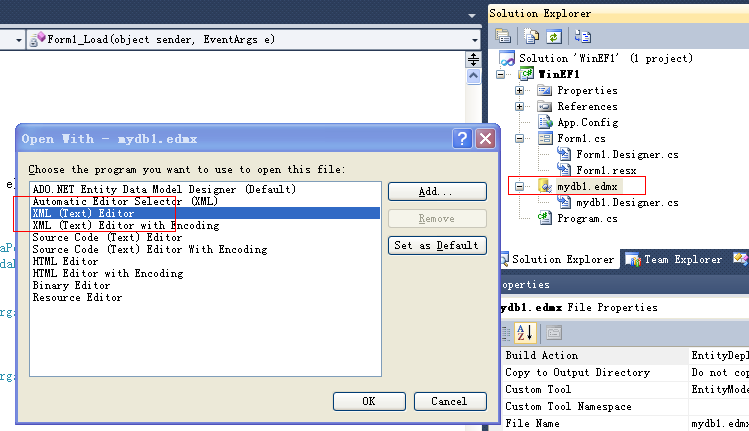
1. 即使数据库端的字段是 Not Null 设置。
2. 所以自动生成的存储模型的表的字段也是 Not Null.
3. 我们手工修改存储模型的该字段 Nullable 属性 = true

缺点：

手工修改会造成不一致，而且更新模型时，会被覆盖掉，维护性差。

另外如果对 people 添加新记录， 则没有country 值。 如果数据库定义为非空，而且没有默认值，写入数据库则出错。

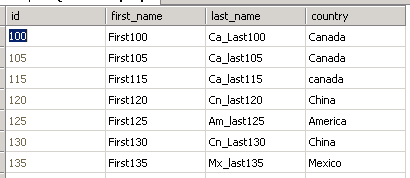
**使用 xml editor 来编辑 xxx.edmx 模型定义文件：**

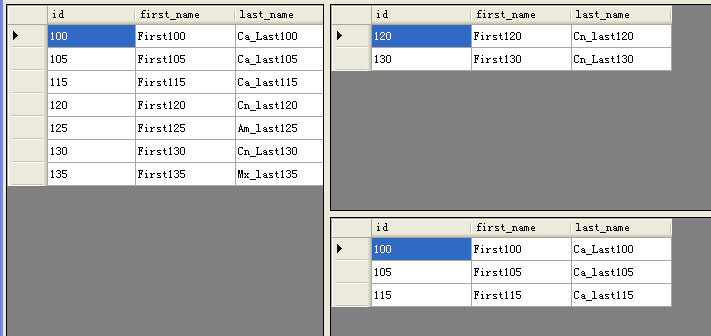
****

然后我们可以看到三个段的XML定义：Storage Models, Conceptual Models, Mapping Models:



数据库的数据记录：





myb = new mydb1();

this.dg1.DataSource = myb.people.ToList();

this.dg2.DataSource = myb.people.OfType<ChinaPeople>().ToList();

this.dg3.DataSource = myb.people.OfType<CanadaPeopel>().ToList();

**注意：** 如果基类people是实体类， 则结果集合是全集。

插入操作：

ChinaPeople cnp = new ChinaPeople

{

first\_name = "ChinaFirst",

last\_name = "ChinaLast"

};

CanadaPeople cap = new CanadaPeople

{

first\_name = "CanadaFirst",

last\_name = "CanadaLast"

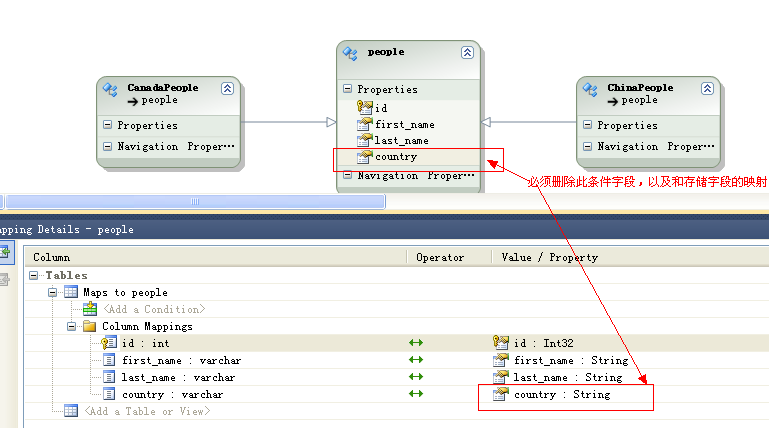
};

this.myb.people.AddObject(cnp);

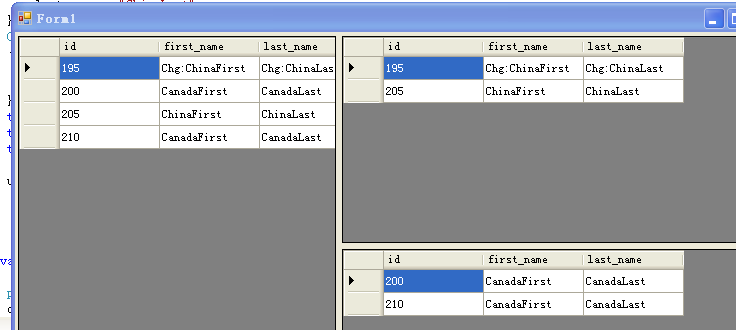
this.myb.people.AddObject(cap);

this.myb.SaveChanges();

1. 如果 people 类设置为抽象类 abstract = true. 那么 people 类不能实例化。只有它的继承类 ChinaPeople, CanadaPeople 类可以实例化。



**注意：** 如果基类people是抽象类（abstract=true）， 则结果集合是所有条件集合的并集。 也是全集的子集。 这里就是 ChinaPeople + CanadaPeople 的并集合。



修改：

people op = this.myb.people.FirstOrDefault();

op.first\_name = "Chg:" + op.first\_name;

op.last\_name = "Chg:" + op.last\_name;

ChinaPeople cnp = this.myb.people.OfType<ChinaPeople>().FirstOrDefault();

cnp.first\_name = "CnChg:" + cnp.first\_name;

cnp.last\_name = "CnChg:" + cnp.last\_name;

CanadaPeople cap = this.myb.people.OfType<CanadaPeople>().FirstOrDefault();

cap.first\_name = "CaChg:" + cap.first\_name;

cap.last\_name = "CaChg:" + cap.last\_name;

this.myb.SaveChanges();

插入： people 作为抽象类，不能被实例化。

ChinaPeople cnp = new ChinaPeople

{

first\_name = "ChinaFirst",

last\_name = "ChinaLast"

};

CanadaPeople cap = new CanadaPeople

{

first\_name = "CanadaFirst",

last\_name = "CanadaLast"

};

this.myb.people.AddObject(cnp);

this.myb.people.AddObject(cap);

this.myb.SaveChanges();

删除：

people op = this.myb.people.FirstOrDefault();

if (op != null)

this.myb.people.DeleteObject(op);

else

MessageBox.Show("Record can not be found");

ChinaPeople cnp = myb.people.OfType<ChinaPeople>().FirstOrDefault();

this.myb.people.DeleteObject(cnp);

CanadaPeople cap = myb.people.OfType<CanadaPeople>().FirstOrDefault(); //Where(p => p.id == 225).SingleOrDefault();

this.myb.people.DeleteObject(cap);

this.myb.SaveChanges();

**4.关于字段的映射问题：各个条件表，可以有自己字段与存储模型对应。**

存储模型的字段，和 （ 条件表 + 基本表 ）的字段来对应。 也必须遵守映射的基本原则：

存储模型的字段不能重复映射到概念模型的字段里。

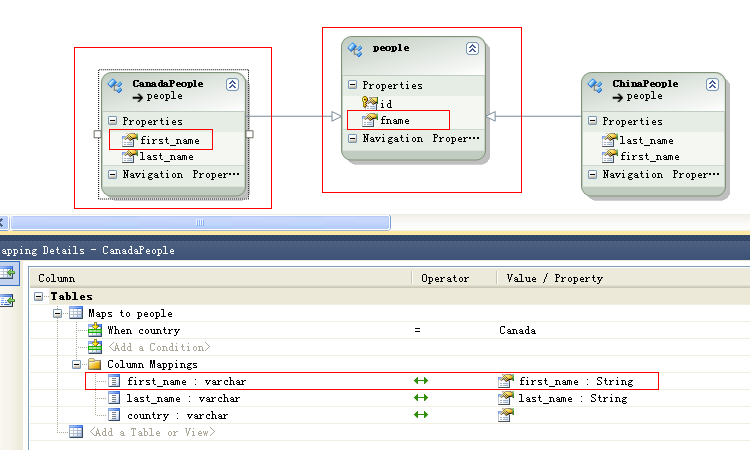
如下则出错：

存储模型：id, first\_name, last\_name, country;

first\_name->people.fname id -> people.id country 条件

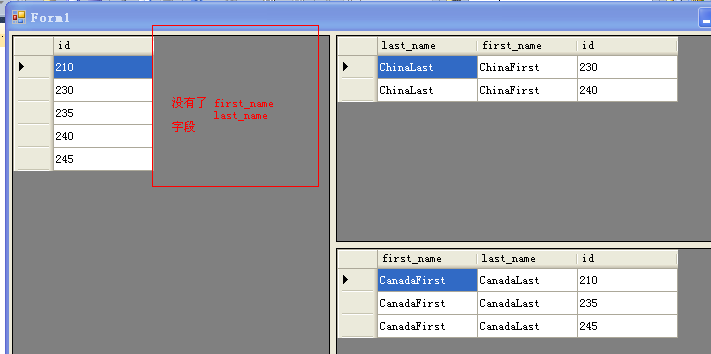
first\_name ->CanadaPeople.first\_name, last\_name->CanadaPeople.last\_name

出错： 存储模型 first\_name 对应了两个概念模型里的字段。 映射原则:一对多不行, 多对一则可以.



根据需要， 必须将 people.fname 删除.

**修正后：**

****

5. 访问的技巧： 在对象容器里， 也不存在 CandaPeople, ChinaPeople 这样的实体对象。 要访问他们， 或者在

他们的范围里操作， 使用如下方法：基表.OfType(TEntity) -> ObjectQuery<TEntity>

ChinaPeople cnp = myb.people.OfType<ChinaPeople>().FirstOrDefault();

this.myb.people.DeleteObject(cnp);

CanadaPeople cap = myb.people.OfType<CanadaPeople>().FirstOrDefault();

this.myb.people.DeleteObject(cap);

cap = myb.people.OfType<CanadaPeople>().Where(p => p.id == 245).SingleOrDefault();

cap.first\_name = "Caca First";

cap.last\_name = "Caca Last";

this.myb.SaveChanges();

**例如：**

ObjectQuery<ChinaPeople> cnp = myb.people.OfType<ChinaPeople>();

string str = null;

foreach (ChinaPeople c in cnp)

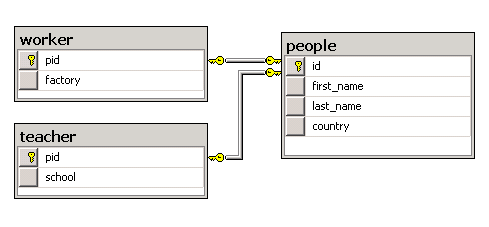
{

str += String.Format("id:{0}-{1}-{2}\n", c.id, c.first\_name, c.last\_name);

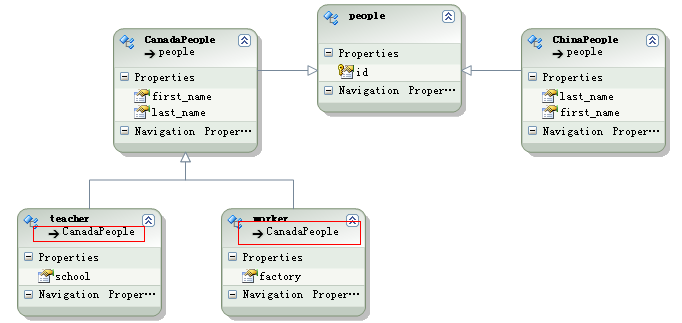
}

MessageBox.Show(str);

**情况三： 多层继承：我们再创建两个表：**

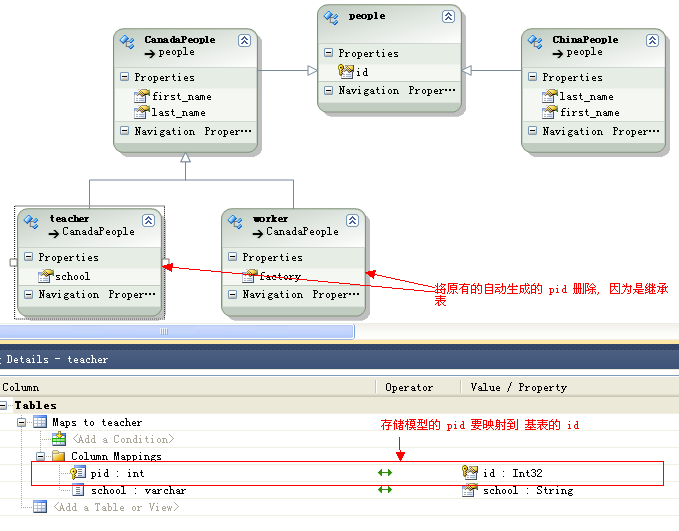
****

**我们把表添加到数据模型里： 1）删除自动定义的 1 对 0..1 的关系. 2) 定义继承关系: 由于条件表格的问题. 所以基表应该选择条件表: 如下图:**

****

**要做几件事情: 1) 删除添加的表的 id. 因为他将继承自基表.**

**2) 而存储模型的表的 pid , 将映射到 基表的 id 上, 这样对于 CRUD 操作, 将自动形成对应.**

****

**插入操作:**

teacher nt = new teacher

{

first\_name = "Thomas",

last\_name = "Yang",

school = "Van School"

}; // country = China 是自动给出的, 因为条件表的原因. id 则是自动给出的.

this.myb.people.AddObject(nt);

this.myb.SaveChanges();

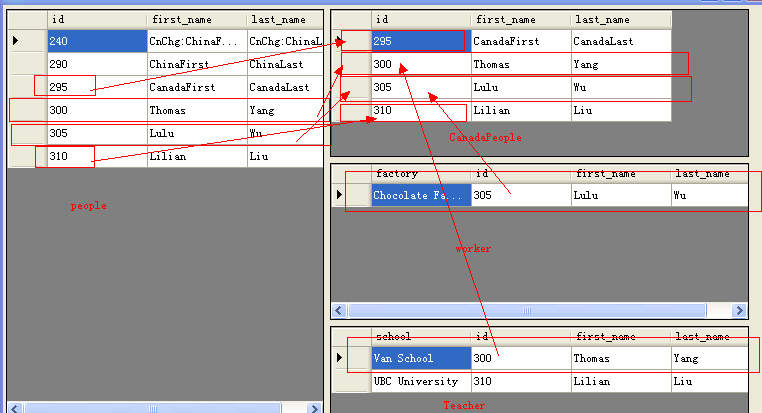
**注意:** 这样同时在两个表插入了数据: people (限定country = Canada) , teacher

**People : id, first\_name, last\_name , country**

****

**Teacher : pid , school**

****

****

**插入操作:**

teacher nt = new teacher

{

first\_name = "Lilian",

last\_name = "Liu",

school = "UBC University"

};

worker nw = new worker

{

first\_name = "Lulu",

last\_name = "Wu",

factory = "Chocolate Factory"

};

this.myb.people.AddObject(nw);

this.myb.people.AddObject(nt);

this.myb.SaveChanges();

**修改操作:**

teacher ut = myb.people.OfType<teacher>().FirstOrDefault();

ut.first\_name = "Christine";

ut.last\_name = "Bai";

ut.school = "BCIT";

this.myb.SaveChanges();

**删除:**

people npp = myb.people.FirstOrDefault();

myb.people.DeleteObject(npp);

this.myb.SaveChanges();

upd();

我们可以通过 people 来删除, 删除基表也会同时删除所有继承表的记录, 不管约束规则如何, 是否级联操作,也会删除..

// 删除继承表, 也会一同删除基表的记录.

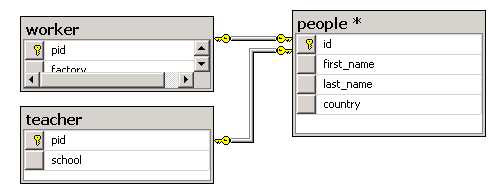
teacher ntt = myb.people.OfType<teacher>().FirstOrDefault();

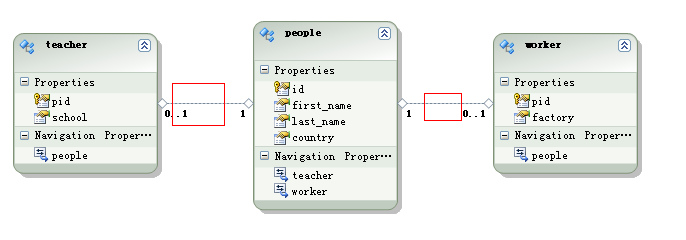
myb.people.DeleteObject(ntt);

this.myb.SaveChanges();

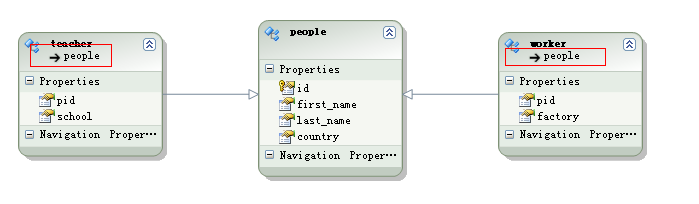
总结,无论通过 基表, 还是继承表删除记录, 都会删除基表和继承表的记录. 一同删除.

**情况四: 不同的表直接继承基表:**

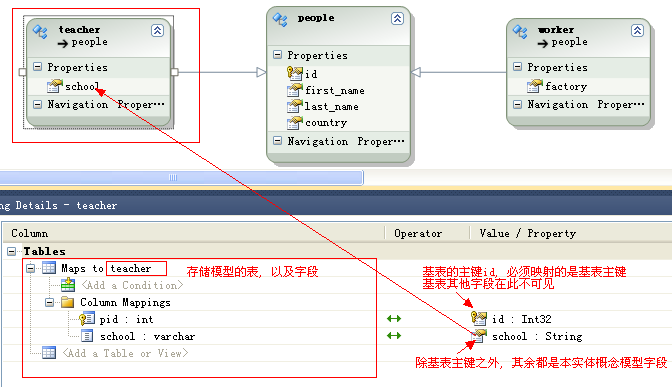
****

****

将自动生成的关系删除: 对继承表指定基表, 在属性 Base Type: people



删除两个继承表的主键, 因为主键继承自基表, 所以不需要. 另外将表映射做修改一下:



为什么继承表的存储模型主键映射的是基表的主键, 这样做就会自动对应关系了.

这样就完成了设置!!

**插入操作:**

people np = new people {

first\_name = "People F:" + DateTime.Now.ToLongTimeString(),

last\_name = "People Last",

country = "China"

};

myb.AddTopeople(np);

teacher nt = new teacher

{

first\_name = "Teacher F:" + DateTime.Now.ToLongTimeString(),

last\_name = "Teacher L:",

country = "Teacher",

school = "B.C. School:" + DateTime.Now.ToLongTimeString()

};

myb.people.AddObject(nt);

worker nw = new worker

{

first\_name = "work f:" + DateTime.Now.ToLongTimeString(),

last\_name = "work l",

country = "Worker",

factory = "Factory:" + DateTime.Now.ToLongTimeString()

};

myb.people.AddObject(nw);

myb.SaveChanges();

因为 people 是实体类, 所以是全集. 包含了 Teacher, Worker 的集合.

**修改操作:**

teacher nt = myb.people.OfType<teacher>().FirstOrDefault();

nt.school = "change school";

nt.first\_name = "change f";

nt.last\_name = "change l";

worker nw = myb.people.OfType<worker>().FirstOrDefault();

nw.factory = "change facotry";

nw.first\_name = "work fn";

nw.last\_name = "work ln";

myb.SaveChanges();

通过继承表修改, 则一起修改 基表和继承表的字段值.

但是通过基表则无法看见继承表的那些字段, 所以无法修改继承表的字段值. 但是可以修改基表的字段值.

**删除操作:**

people np = myb.people.FirstOrDefault();

myb.people.DeleteObject(np);

myb.SaveChanges();

删除基表则连同继承表对应的记录一同删除, 不管约束设置如何, 级联删除与否, 都一同删除.

teacher np = myb.people.OfType<teacher>().FirstOrDefault();

myb.people.DeleteObject(np);

myb.SaveChanges();

删除继承表, 也会将基表的记录一并删除.

**基表为抽象表: abstract = true ;**

则 people 的集合是 teacher + worker 的并集.

people 是抽象类, 不能实例化 people , 通过people.AddObject() 添加.

但是可以通过抽象类来删除,修改:

people np = myb.people.FirstOrDefault();

np.first\_name = "change from people";

myb.SaveChanges();

people np = myb.people.FirstOrDefault();

myb.people.DeleteObject(np);

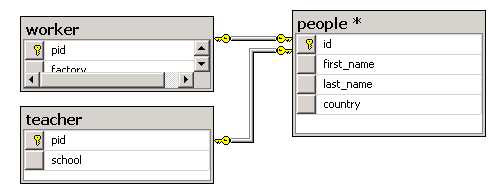
myb.SaveChanges();

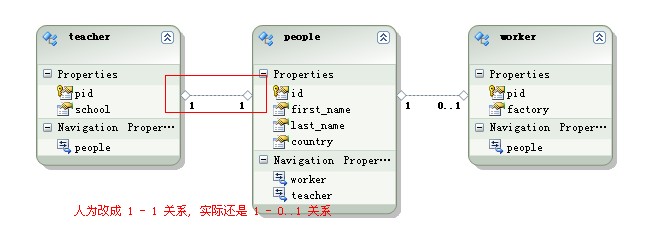
* **关系**

**情况一: 一对一 关系: 即 1 <-> 0..1 或者 1 <-> 1 (其实不存在, 还是有主表与附表之分)**

因为在数据模型里, 必须有外键约束才能建立关系. 实际还是有主次之分的.

所以 1 对 1 关系, 还是 1 对 0..1 的关系.





**插入操作:**

通过主表, 将附表添加.

people np1 = new people

{

first\_name = "Maxwell",

last\_name = "Liu:" + DateTime.Now.ToLongTimeString(),

country = "CHN"

};

teacher nt = new teacher

{

pid = 500,

school = "Camb:" + DateTime.Now.ToLongTimeString()

};

np1.teacher = nt; //通过主表 添加 附表

myb.people.AddObject(np1);

myb.SaveChanges();

通过附表将主表添加:

people np1 = new people

{

first\_name = "Maxwell",

last\_name = "Liu:" + DateTime.Now.ToLongTimeString(),

country = "CHN"

};

teacher nt = new teacher

{

pid = 500,

school = "Camb:" + DateTime.Now.ToLongTimeString()

};

nt.people = np1;

myb.teacher.AddObject(nt);

或者这样也可以:

nt.people = np1;

myb.people.AddObject(np1);

//myb.teacher.AddObject(nt);

myb.SaveChanges();

**修改操作**:

同时修改主表和附表的值:

people np = myb.people.FirstOrDefault();

np.first\_name = "Hello world";

np.teacher.school = "Hello School";

myb.SaveChanges();

也可以通过赋予新对象更改附表:

people np = myb.people.FirstOrDefault();

np.first\_name = "Hello world";

teacher nt = new teacher

{

school = "chage: school"

};

np.teacher = nt;

myb.SaveChanges();

是否能够把附表的记录移动关系, 属于这条记录的附表记录, 转移那条记录, 成为那条记录的附表记录.

people np = myb.people.FirstOrDefault();

np.first\_name = "Hello world";

teacher nt = myb.teacher.Where(p=>p.pid==520).SingleOrDefault();

np.teacher = nt;

myb.SaveChanges();

**注意:** 这是不可以的. 移动附表记录的关系是不可以的. 可以删除, 重新赋值.

修改注意:

teacher nt = new teacher

{

pid = 550,

school = "Change:" + DateTime.Now.ToLongTimeString()

};

myb.people.Where(p=>p.id==550).Single().teacher = nt;

这样修改是可以的, 刚好 id 都是 550

teacher nt = new teacher

{

pid = 550,

school = "Change:" + DateTime.Now.ToLongTimeString()

};

myb.people.Where(p=>p.id==555).Single().teacher = nt;

这样修改是不可以的. 出错!! id = 555 , 和 pid = 550

当然如果不指定 id , 也是可以的. 有则改之, 无则添加.

**删除操作:**

删除附表:

people np = myb.people.FirstOrDefault();

myb.teacher.DeleteObject(np.teacher);

也可以:

people np = myb.people.FirstOrDefault();

np.teacher = null;

myb.SaveChanges();

people np = myb.people.FirstOrDefault();

myb.DeleteObject(np.teacher); // 不同哦, 使用的是数据容器

myb.SaveChanges();

删除主表, 不管数据库,数据模型的约束规则如何, 是否级联删除. 都将删除附表的记录. 联级删除.

people np = myb.people.FirstOrDefault();

myb.DeleteObject(np);

myb.SaveChanges();

people np = myb.people.FirstOrDefault();

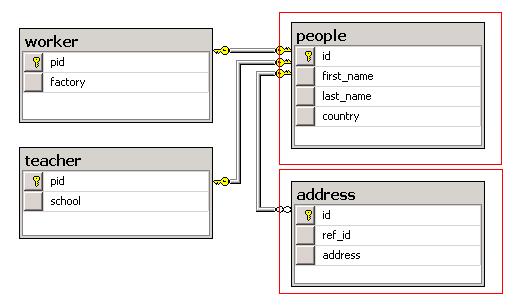
myb.people.DeleteObject(np); // 使用数据容器的删除方法.

myb.SaveChanges();

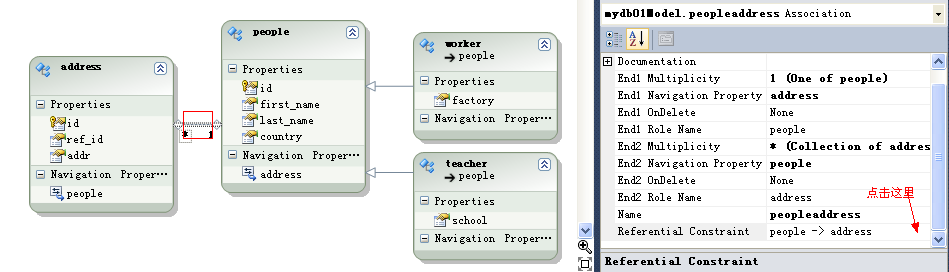
**情况二: [(1)—(\*)] 关联：**

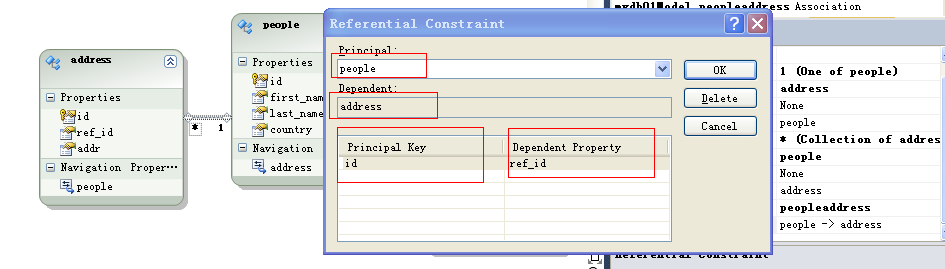
如果主键设置为 identity , 则会将空值传递给数据库, 数据自动生成id值, 再返回给值该客户端实体的相关字段.

1) 必须有外键约束: 主表对附表 一对多 关系.



在概念模型里: 必须对关联设置正确的映射:





**插入新记录:**

DataGridView 的click事件获取id 值:

private void dg1\_CellClick(object sender, DataGridViewCellEventArgs e)

{

this.sid = (int)this.dg1.Rows[e.RowIndex].Cells[0].Value;

this.dg3.DataSource = myb.people.Where(p => p.id == sid).SingleOrDefault().address.ToList();

}

-------------------------------------------------------------------------------------------------------

可以通过归于主表来添加:

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

address naddr = new address

{

addr = "Burnaby:" + DateTime.Now.ToLongTimeString()

};

naddr.people = np;

myb.address.AddObject(naddr);

myb.SaveChanges();

-------------------------------------------------------------------------------------------------------

可以通过主表来添加:

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

address naddr = new address

{

ref\_id = 722,

addr = "Burnaby:" + DateTime.Now.ToLongTimeString()

};

np.address.Add(naddr);

myb.SaveChanges();

注意: 有趣的事情, 主表主键id不存在722的值, ref\_id=722 , 也准确添加, 因为ref\_id 的值是取主键id

-------------------------------------------------------------------------------------------------------

可以通过继承表来添加

address naddr = new address

{

addr = "Burnaby:" + DateTime.Now.ToLongTimeString()

};

teacher nt = new teacher

{

first\_name = "Lilian",

last\_name = "Liu",

school = "UBC",

country = "Canada"

};

nt.address.Add(naddr);

myb.people.AddObject(nt);

myb.SaveChanges();

-------------------------------------------------------------------------------------------------------

直接添加, 前提是已经知道主表的id值, 否则违反约束出错.

address naddr = new address

{

ref\_id = 722,

addr = "Burnaby:" + DateTime.Now.ToLongTimeString()

};

myb.address.AddObject(naddr);

myb.SaveChanges();

如果主表没有id=722, 则出错, 有则添加.

-------------------------------------------------------------------------------------------------------

**修改以及转移:**

将挂在 id=730的记录下的所有 address 子表的记录, 转移到另外的 id 下

people op = myb.people.Where(p => p.id == 730).SingleOrDefault();

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

//np.address.Clear();

while (op.address.FirstOrDefault() != null)

{

np.address.Add(op.address.FirstOrDefault()); // 使用Add添加, 会将旧主抛弃掉, 投入新主怀抱.

}

myb.SaveChanges();

this.dg3.DataSource = myb.people.Where(p => p.id == sid).SingleOrDefault().address.ToList();

//此代码会有问题: 因为 op.address 被添加到其他 id 下之后, 就从原处删除了. 附加到其他id 下了.

所以 op.address 集合的大小是动态变化的. 而完美的代码如上 使用 while 判断什么时候为空.

foreach (address addr in op.address)

{

np.address.Add(addr);

}

转移也可以通过直接修改子表的外键: ref\_id

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

while( np.address.FirstOrDefault() != null )

{

address addr = np.address.FirstOrDefault();

addr.ref\_id = 740; // 修改外键, 此外键必须存在, 否则违反外键约束出错.

}

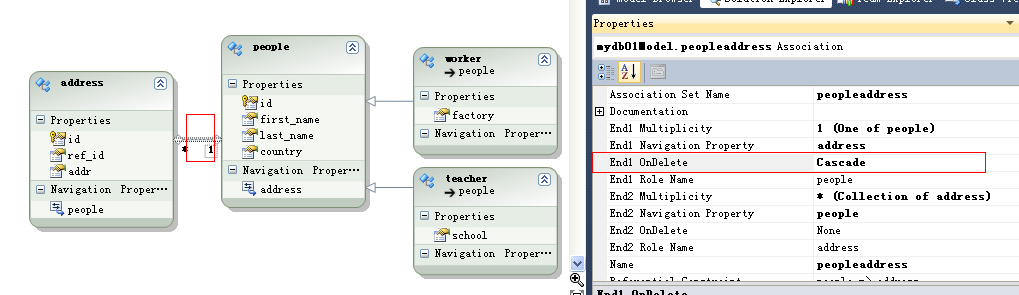
myb.SaveChanges();

**删除操作**

**---------------------------------------------------------------------------------------------------------------------------------------**

删除主表记录:

不管数据库端的实际删除规则如何, 主要取决于Entity Model 的删除规则:



people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

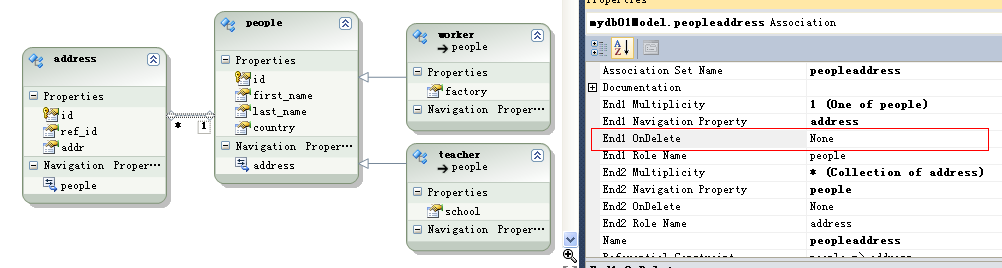
myb.DeleteObject(np);

myb.SaveChanges();

如果子表有记录, 则连同子表的相关集合一起删除.

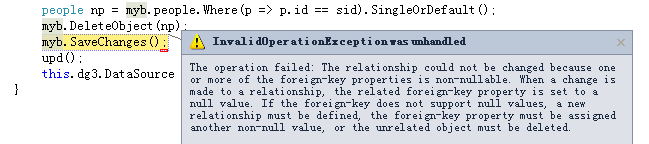
-------------------------------------------------------------------------------------------------------------------------------------------------

如果 Entity Model 的删除规则: 是 None

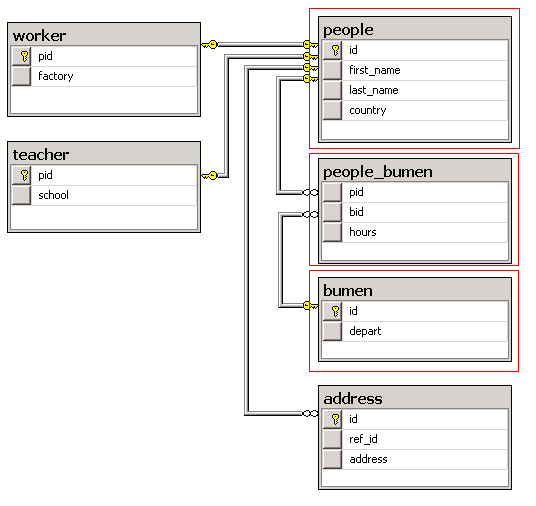


如果子表没有相关集合, 这直接删除主表的记录.

如果子表有相关的集合, 则提示违反了外键约束出错!.

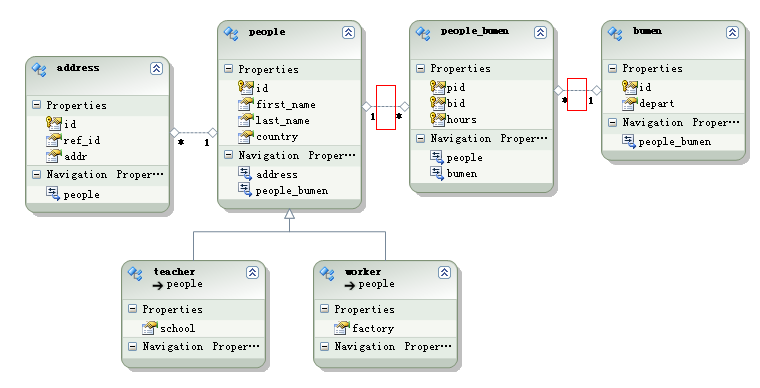


**情况三: [(\*)—(\*)]关联：**

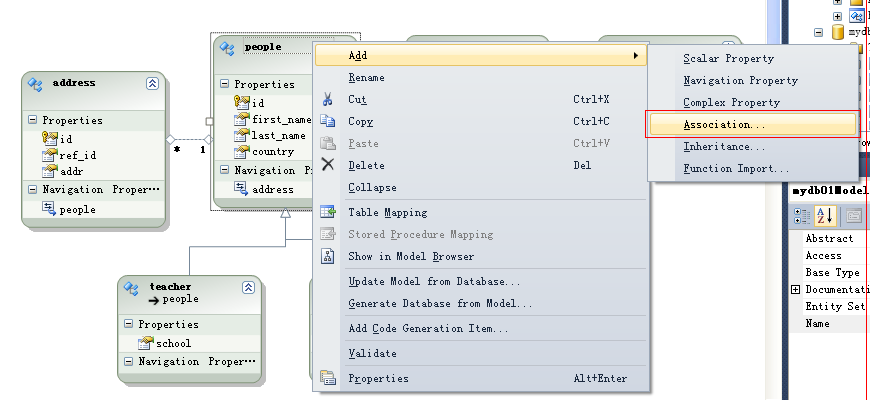


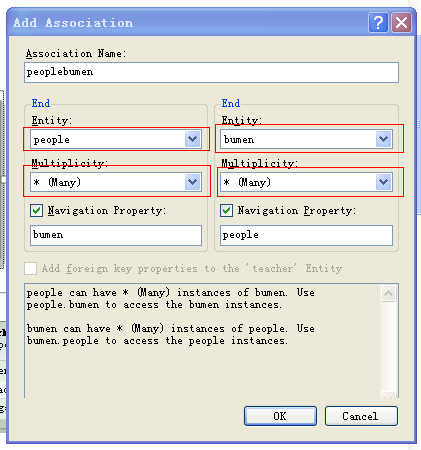
**关系**: 人属于多个部门, 部门可以有多个人; 这是多对多关系.

需要删除自动生成的一对多关系:



添加新的关系:



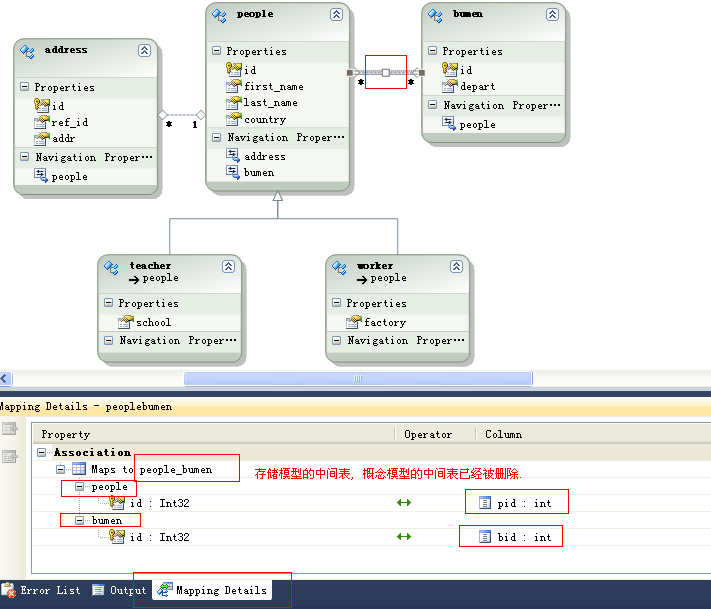


接着重新对关联定义映射关系:

1) 将概念模型的存储表删除, 记得存储模型的中间表一定要保留.

2) 将关联的映射到存储模型的中间表. 这时对多对多的两个关联表的主键进行中间表映射.

如下图:



这样多对多关联就完成了!

**插入操作:**

往已有的people 里新增 bumen:

bumen nbu = new bumen

{

depart = "Teaching:" + DateTime.Now.ToLongTimeString()

};

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

-----------------------------------------------

// 通过部门来添加已存在的人

nbu.peoples.Add(np);

------------- 或者 -----------------------------

//通过已存在的人添加部门

np.bumens.Add(nbu);

------------- 或者 -----------------------------

//通过相互添加,都可以

nbu.peoples.Add(np);

np.bumens.Add(nbu);

myb.SaveChanges();

往已存在的人里添加已存在的部门; 而其该部门本来就属于自己.

此时是并不会添加新记录, 而是修改.

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

bumen nbu = np.bumens.FirstOrDefault(); // 注意部门本来属于自己

nbu.depart = "change bumen"; // 修改了部门的信息

np.bumens.Add(nbu); // 自己添加本来属于自己的部门

myb.SaveChanges(); // 并不增加新记录, 相当于修改操作.

往已存在的人里添加属于 ”别人的部门”, 则部门表不变, 中间表自己新增部门, 而不会把别人的转移过来. 别人的维持不变

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

// 找到属于别人的部门

bumen nbu = myb.people.Where(p => p.id == 740).SingleOrDefault().bumens.FirstOrDefault();

nbu.depart = "change:" + nbu.depart;

np.bumens.Add(nbu); // 把属于别人的部门添加到自己的部门列表.

myb.SaveChanges(); // 有则改之, 无则添加.

**总结:** 只要是添加已存在的部门, 没有转移的问题, 有则改之, 无则添加.

人也是新的, 部门也是新的: 关系弄好后, 只需要添加人, 或者部门到实体里即可. 二者选一个即可.

people np = new people

{

first\_name = "Zhongguoren",

last\_name = "China",

country = "China"

};

bumen nb = new bumen

{

depart = "Production"

};

nb.peoples.Add(np); // 建立关系

myb.bumen.AddObject(nb); // 只需要添加部门即可, 或者添加人也可以. 二选一.

myb.SaveChanges();

如果关系建立后, 人也添加, 部门也添加到实体. 则不会把人重复添加多一次. 因为是对象已存在. 不用担心记录重复.

myb.bumen.AddObject(nb);

myb.people.AddObject(np);

**修改操作:**

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

np.first\_name = "world";

bumen nb = np.bumens.FirstOrDefault();

nb.depart = "good";

myb.SaveChanges();

**删除操作:**

删除人, 连同人所属于的部门也删除, 并非是删除部门表, 注意而是删除中间表:

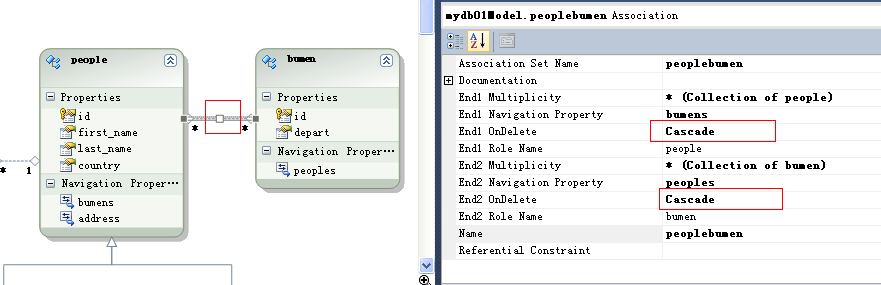
注意这种删除不需要删除级联. 设置级联删除反而出错. 注意主表删除连带删除的是中间表.

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

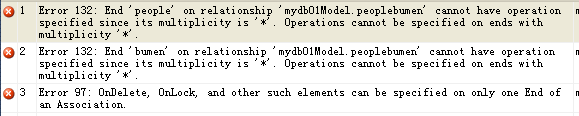
myb.DeleteObject(np);

myb.SaveChanges();

注意多对多关联, 删除规则不需要设置成为: 级联. 如果设置成级联则出错.



设置级联会出错: 如下:



单个删除:

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

np.bumens.Remove(np.bumens.FirstOrDefault()); // 注意判断非空.

myb.SaveChanges();

删除所有:

people np = myb.people.Where(p => p.id == sid).SingleOrDefault();

np.bumens.Clear(); // 删除所有

myb.SaveChanges();

ADO.NET Entity Framework 执行SQL Store procedure 的注意问题：

## Using Stored Procedures in the Entity Framework with Scalar Return Values

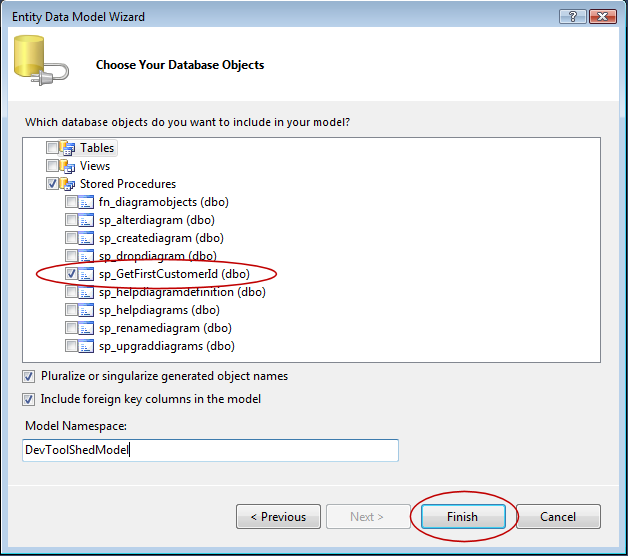
* [entity framework](http://www.devtoolshed.com/category/tags/entity-framework)
* [stored procedure](http://www.devtoolshed.com/category/tags/stored-procedure)

Although stored procedures may be on their way out (see post here: <http://stackoverflow.com/questions/216569/are-the-days-of-the-stored-procedure-numbered>), they are still supported in Entity Framework and can be used relatively easily. There are many tutorials on how to use a stored procedure in the Entity Framework to return a set of data and bind it to an Entity. But I had a little more trouble figuring out how to call a stored procedure that returns a scalar value. This tutorial shows the basics for calling a stored procedure using the Entity Framework.

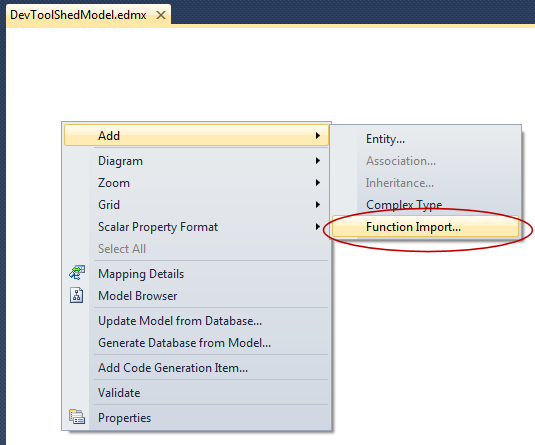
You can download the sample code using the link at the bottom of this tutorial.

First, run the "generate\_schema.sql" script in the EntityFrameworkStoredProcedure.Data project to create your database schema for the sample and generate a stored procedure. I have a really simple stored procedure that just returns the primary key of the first row in the sample Customer table.

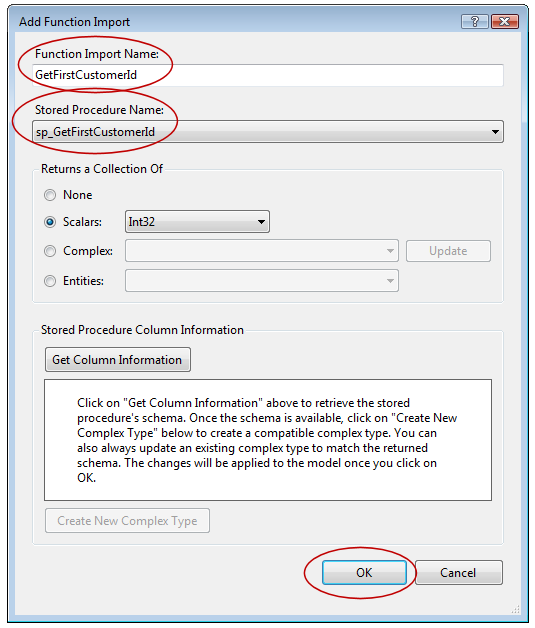
If you are following along and want to get this set up, here are the steps to get your own project set up using a stored procedure in the Entity Framework. Create your new Entity Framework (.edmx) file in Solution Explorer. The sample stored procedure we are using is called "sp\_GetFirstCustomerId". Now, when you update the database model, you are presented with a dialog to choose the database items to add to your model.



Once you click Finish, the model will be read and the dialog will close. Now, you might think, what the heck? Nothing seems to happen. But basically here you have just told Entity Framework about your stored procedure. But to actually get it hooked up to the context so you can call it, you need to Import the stored procedure as a Function. Right-click on the workspace area of your Entity model and choose Add -> Function Import.



In the Add Function Import dialog, enter the name you want your stored procedure to be referred to in your model, choose your procedure from the drop down list, and choose the return value of the procedure to be Scalar in this case.



Now you can call this method using code like this:

// create new instance of the context

Data.devtoolshedEntities context = new Data.devtoolshedEntities();

// call the stored procedure to get the return value

System.Nullable<int> iReturnValue = context.GetFirstCustomerId().SingleOrDefault();

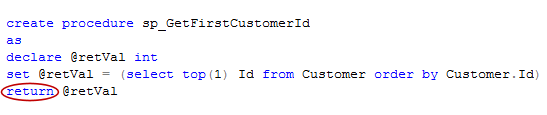
if (iReturnValue.HasValue)

{

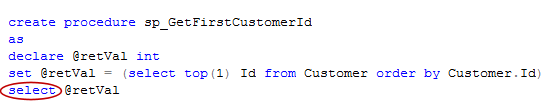
// return was successful!

}

Now, if you run this as is, it all should work and get your return value fine BUT if you look closely, you will see that I have made a key change to my stored procedure. Originally, I used the keyword "return" in the stored procedure to return the value from the function but after some pain and suffering, it turns out that Entity Framework cannot support Stored Procedure Return scalar values out of the box. BUT there is a work around. For instance, here is how I originally declared my stored procedure. Notice the "Return" keyword at the bottom.



To get this to work with Entity Framework, you need to use "Select" instead of "Return" to return back the value. This will return 0 or 1 nullable rows as a side effect so make sure to check for that but it will also make it so that Entity Framework can actually call your procedure without an error. Here is the updated stored procedure declaration using "select".



You can download the code for this project here.

ADO.NET Entity Framework - Store Procedure 返回多个结果集的问题：

EF不支持存储过程返回多表联合查询的结果集。

ALTER PROCEDURE [dbo].[sp\_current\_config]

@config\_id int output

AS

BEGIN

SET NOCOUNT ON;

select @config\_id = max(id) from app\_config;

SELECT \* FROM app\_config;

SELECT \* FROM info\_system;

END

LOG\_DB db = new LOG\_DB(sql.EntityString("LogDB4"));

ObjectParameter p1 = new ObjectParameter("config\_id", 0);

ObjectResult<LOG\_CONFIG>res = db.sp\_current\_config(p1);

ObjectResult<LOG\_SYSTEM> res1 = res.GetNextResult<LOG\_SYSTEM>(); - 虽然方法存在，可是返回 null

注意：res1 返回 null, 所以 res1.FirstOrDefault() 会抛出运行错误。

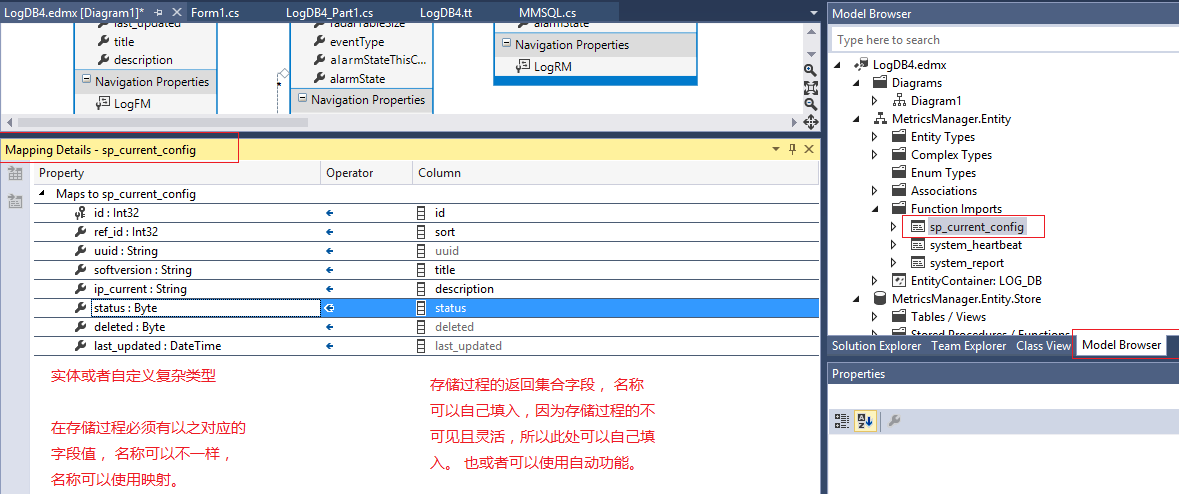
MessageBox.Show(string.Format("Count: {0} - {1}", res.FirstOrDefault().file\_path, p1.Value));

解决方案：

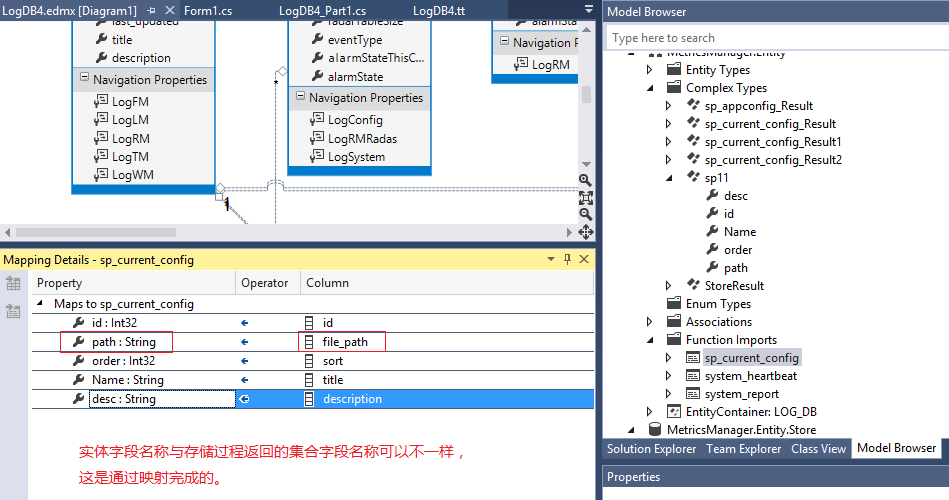
关于存储过程返回数据集合的字段映射问题：

1） 如果Entity Framework 实体字段太多， 而存储过程返回的字段太少， 则出错。

实体的字段或者自定义复杂类型的字段，必须全部得到映射关系。 否则出错



映射关系很重要， 实体名称和实际数据库存储过程返回字段的名称可以不一样， 这是通过映射完成的。



2) 数据存储过程的返回的字段可以比实体字段要多。 也就是实体字段是返回结合的子集合（针对字段数量）。

ALTER PROCEDURE [dbo].[sp\_current\_config]

@config\_id int output

AS

BEGIN

SET NOCOUNT ON;

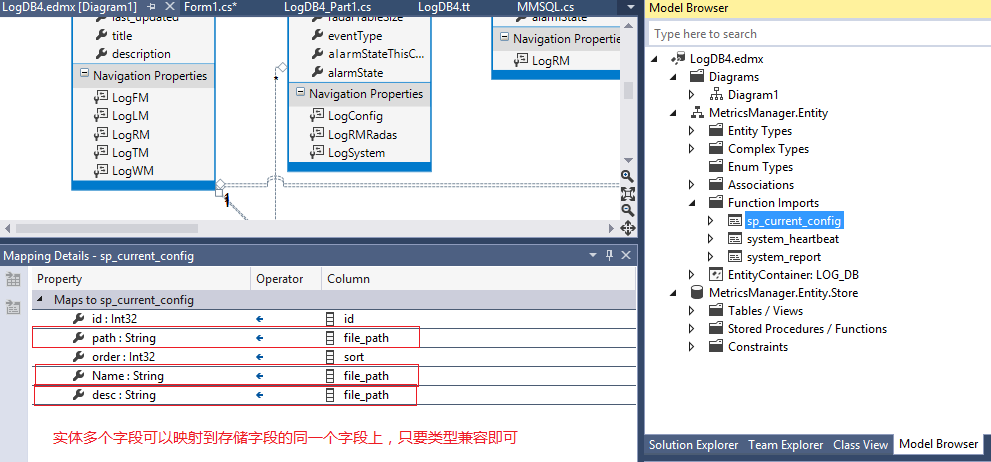
select @config\_id = max(id) from app\_config;

SELECT id, title, description, file\_url, status, file\_path, sort FROM app\_config;

SELECT \* FROM info\_system;

END

3） 实体多个字段可以映射到同一个存储字段上， 只要类型兼容即可。



LOG\_DB db = new LOG\_DB(sql.EntityString("LogDB4"));

ObjectParameter p1 = new ObjectParameter("config\_id", 0);

ObjectResult<sp11> res = db.sp\_current\_config(p1);

ObjectResult<LOG\_SYSTEM> res1 = res.GetNextResult<LOG\_SYSTEM>();

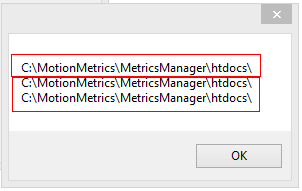
sp11 sp\_res = res.FirstOrDefault();

MessageBox.Show(string.Format("{0}\n{1}\n{2}",

sp\_res.path,

sp\_res.desc,

sp\_res.Name));



MessageBox.Show(string.Format("Count: {0}\n{1}\n{2}",

res.FirstOrDefault().path,

res.FirstOrDefault().desc, - 因为是 SqlReader Move Forward

res.FirstOrDefault().Name));

不能多次调用 FirstOrDefault() , 因为是 SqlReader, 不能回到第一条记录。 只能执行一次

这样也不可以

sp11 sp\_res = res.FirstOrDefault();

sp\_res = res.Where(p => p.id == 3).FirstOrDefault();

这样也不可以

sp11 sp\_res = res.Where(p => p.id == 2).FirstOrDefault();

sp\_res = res.Where(p => p.id == 3).FirstOrDefault();

这样扫描是可以的：

string str = string.Empty;

foreach (sp11 el in res)

{

str += string.Format("Path: {0}\n", el.path);

}

MessageBox.Show(str);