模型（2）

1.模型对应关系

关系数据库最强大的地方在于“关系”，也即表和表之间是有关联的，这种关联有三种类型：

- 一对一
- 一对多
- 多对多

1.1 一对一

一个学生有一个档案，一个档案属于一个学生，那么学生表和档案表就是一对一关系。学生表是主表，档案表是从表，从表中有一个外键和学生表关联，并且要求外键取值唯一。对应关键字为：OneToOneField

```
#创建模型
class Student(models.Model):
    sno = models.CharField(max_length=6,primary_key=True)
    sname = models.CharField(max_length=100,null=False)
    ssex = models.CharField(max_length=2,default='男',null=True)
    sage = models.IntegerField(null=True)
    sclass = models.CharField(max_length=10,null=True)

    def __str__(self):
        return "no:{},name:{}".format(self.sno,self.sname)

    class Meta:
        db_table = 'student'

class Archives(models.Model):
    idcard = models.CharField(max_length=18, unique=True)
    address = models.CharField(max_length=200,null=True)
    # on_delete=models.CASCADE 级联删除，删除学生会连同档案一块删除
    student = models.OneToOneField(Student, on_delete=models.CASCADE)

    def __str__(self):
```

```
        return "{},{}".format(self.idcard,self.address)

    class Meta:
        db_table = 'archives'
```

- 增加数据

```
def addstudent(request):
    student = Student()
    student.sno = '180502'
    student.sname = '杨康'
    student.sage = 22
    student.save()
    return HttpResponse("增加了一个学生")

def addarchives(request):
    stu = Student.objects.get(pk='180503')
    arc = Archives()
    arc.idcard = '130098384893838953'
    arc.student = stu   #学生对象必须已经保存到数据库，否则错误
    arc.save()
    return HttpResponse("增加档案")
```

- 删除数据

```
def deletestudent(request):
    student = Student.objects.get(pk='180503')
    student.delete()
    return HttpResponse("删除学生")
```

- 正向查询

```
def findstudent(request):
    # 获取学生信息
    student = Student.objects.first()
    print(student)
    # 通过学生对象获取档案信息
    archive = student.archives
    print(archive)
    return  HttpResponse(student)
```

- 反向查询

```
def findarchives(request):
    #获取档案记录
    archive = Archives.objects.first()
    #通过档案获取关联学生信息
    student = archive.student
    return HttpResponse(student)
```

- 跨关系查询

```
def lookup(request):
    #根据档案查学生
    # student = Student.objects.get(archives__pk=1)
    student =
Student.objects.get(archives__idcard='13009488384383838')
    #根据学生查档案
    archive = Archives.objects.get(student__sno='180501')
    return HttpResponse(archive)
```

- on_delete
  - CASECADE 默认，默认级联删除数据
  - PROTECT 保护模式，当从表中存在级联记录的时候，删除主表记录会抛出保护异常，从表中不存在级联数据的时候，是允许删除的
  - SET_XXX
    - NULL 外键字段本身必须允许为空
    - DEFAULT 外键字段本身有默认值
  - DO_NOTHING 什么都不做

1.2 一对多

一个出版社可以出版多本书，一本书只能被一个出版社出版。出版社和图书表属于一对多，一对多一般将主表中的主键并到从表中做外键。在模型中用ForeignKey表示多对一

```
class Publisher(models.Model):
    pname = models.CharField(max_length=100,null=True)

    def __str__(self):
        return self.pname
```

```
    class Meta:
        db_table = 'publisher'

class Book(models.Model):
    bname = models.CharField(max_length=200,null=True)
    #多对一模型通过ForeignKey表示多对一
    #如果publisher定义在book之后，第一个参数应该用字符串'Publisher'
    publisher = models.ForeignKey(Publisher,on_delete=models.CASCADE,
                                  null=True,
                                  db_column='pid',  #表中字段名
                                  related_name='books')  #通过出版社
查图书时使用的关系名

    def __str__(self):
        return self.bname
    class Meta:
        db_table = 'book'
```

- 增加

```
pub = Publisher.objects.first()
pub.books.create(bname='红岩')
book = Book.objects.get(pk=1)
pub.books.add(book)  #book必须已经保存到数据库
books = Book.objects.filter(pk__lt=5)
pub.books.bulk_create(list(books))
```

- 删除和更新

```
pub = Publisher.objects.first()
pub.books.all().delete()  #删除出版社出版的所有图书
pub.books.all().update(bname='ddd')
```

- 正向查询

```
def findpublisher(req):
    pub = Publisher.objects.first()
    print(pub)
    # pub = Publisher()
    book = pub.book_set.all()
    print(book)
    return HttpResponse("查询出版社")
```

- 反向查询

```
def findbook(req):
    book = Book.objects.first()
    return HttpResponse(book.publisher.pname)
```

- 跨关系查询

```
def loopup(req):
    # 根据图书获取出版社
    pub = Publisher.objects.get(book__bname='花样年华927937')
    print(pub)

    # 根据出版社获取图书
    books = Book.objects.filter(publisher__pname='科技出版社5829')
    print(books)
    return HttpResponse("跨关系查询")
```

1.3 多对多

一个买家可以购买多件商品，一件商品可以被多个买家购买，买家和商品之间构成多对多关系，多对多关系必然会生成一张中间表：买家-商品表，记录商品和买家的关系，该表包含商品表主键和买家表的主键

```
from django.db import models

# Create your models here.
class Buyer(models.Model):
    bname = models.CharField(max_length=30)
    level = models.IntegerField(default=1)
```

```
class Goods(models.Model):
    gname = models.CharField(max_length=100)
    price = models.FloatField()
    buyer = models.ManyToManyField(Buyer)  #这种写法自动生成第三张表，但
我们无法直接控制
    def __str__(self):
        return self.gname +"  "+ str(self.price)

#手动创建中间表
class Orders(models.Model):
    buyer =
models.ForeignKey(Buyer,on_delete=models.CASCADE,db_column='bid')
    goods =
models.ForeignKey('Goods',on_delete=models.CASCADE,db_column='gid')
    num = models.Integer(default=1)

class Goods(models.Model):
    gname = models.CharField(max_length=100)
    price = models.FloatField()
    buyer = models.ManyToManyField(Buyer,through='Orders')
```

- 购买商品

```
def sellgoods(req):
    goods = Goods.objects.get(pk=randint(1,Goods.objects.count()))

goods.buyer.add(Buyer.objects.get(pk=randint(1,Buyer.objects.count())
))
    goods.save()
    return HttpResponse("剁手成功")
```

- 删除商品

```
 buyer = Buyer.objects.get(pk=13)
goods = Goods.objects.filter(pk__lt=10)
buyer.goods_set.clear()  #删除所有商品
buyer.goods_set.remove(Goods.objects.get(pk=2))  #删除指定商品
```

- 正向查询

```
def findgoods_by_buyer(req):
    buyer = Buyer.objects.get(pk=13)
    res = buyer.goods_set.all()
    print(res)
    return HttpResponse("由买家查询商品")
```

- 反向查询

```
def findbuyer_by_goods(request):
    goods = Goods.objects.last()
    buyer = goods.buyer.all()
    print(buyer)
    return HttpResponse("由商品查询买家")
```

2.模型继承

django中的数据库模块提供了一个非常不错的功能，就是支持models的面向对象，可以在models中添加Meta，指定是否抽象，然后进行继承

```
class Animal(models.Model):
        xxx
        class Meta:
            abstract = True/False

class Dog(Animal):
        xxx
```

默认模型就是允许继承的，但是默认的继承处理方式不是很合理：

- 默认在父类中定义的字段会存在父类的表中，子类的数据通用部分会存在父表中，子类特有数据会在子表中，子类通过外键进行级联
- 默认方式比较垃圾，效率比较低

开发中，需要将父类抽象化，在元信息中使用abstract=True

- 抽象化的父类不会再数据库生成表了
- 子类会将父类中的通用数据，复制到子表中