# 第十章 Django数据库操作

本节所讲内容：

10.1 ORM概念

10.3 ORM数据库建模部分

10.4 ORM数据库查询部分

10.5 ORM数据库关系部分

## 10.1 ORM概念

到目前为止，当程序涉及到数据库相关操作时，一般都会这么搞：

1、创建数据库，设计表结构和字段

2、使用 MySQLdb(pymysql) 来连接数据库，并编写数据访问层代码

3、业务逻辑层去调用数据访问层执行数据库操作

这种操作并不是不好，但是不得不承认，这样的操作对开发的数据库知识还是有一定的要求的，而且加入因为业务需求，需要切换数据库类型，mysql🡪oracle那么那些查询语句的修改将会是一个繁重的工作，在这种前提下我们引进了ORM数据库操作：

ORM：Object Relational Mapping(关系对象映射)

Django的orm操作本质上会根据对接的数据库引擎，翻译成对应的sql语句；所有使用Django开发的项目无需关心程序底层使用的是MySQL、Oracle、sqlite....，如果数据库迁移，只需要更换Django的数据库引擎即可；ORM主要分为两大部分：

数据库建模部分

数据库查询部分

关于这个orm数据关系型映射我们已经在第一阶段的peewee中有所涉及，在这里就一掠而过了。

## 10.3 ORM数据库建模部分

学习ORM建模之前，首先了解一下数据库建模。

数据库建模就是在设计[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)时，对现实世界进行分析、抽象、并从中找出内在联系，进而确定数据库的结构，这一过程就称为数据库建模。它主要包括两部分内容：确定最基本的数据结构；对约束建模。

在这里需要注意的两个点：

对业务实体的描述

对业务实体间关系的描述

比如说：对学校管理平台进行建模，其中要用数据库来描述 学生 老师 学科 同样也要描述出三者间的关系，

一个学生可以学习多个学科

一个学科可以有多个学生学习

一个学生可以有多个老师

一个老师可以有多个学生

一个老师可以教一个学科

一个学科可以有多个老师

一个老师在一个学科当中有多个学生

一个学生在一个学科当中有一个老师

等等

只有这些关系在数据库当中都表述清楚，那么网站才可以清楚的展示出：每个学员对应的学科和老师、每个老师对应的学科和学员、等等更多业务逻辑。

现在我们可以进行数据库建模了：

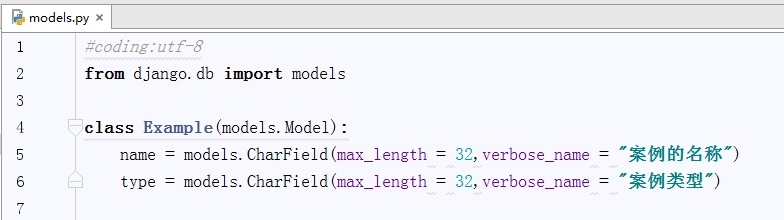
1.首先数据库建模需要对业务逻辑进行分析和规划，思路不够成熟，千万不要下手，再牛的开发，如果数据库搭建的不科学，项目也会go die，所以在开发之前，开发者需要对开发网站的需求进行分析，然后定义合理的数据库，这才是正常的开发流程

2.然后根据分析描述数据库，django当中描述数据库如下

类名对应 表名

类变量对应 字段名

标准格式如下：



from django.db import models

class Example(models.Model):

name = models.CharField(max\_length = 32,verbose\_name = "案例的名称")

type = models.CharField(max\_length = 32,verbose\_name = "案例类型")

这里的例子需要注意以下几点：

1. django的模型的类的名称首字母大写
2. django的模型的类必须继承models.Model
3. django模型的models类当中有定义好的字段类型，上述是字符串类型，类型有好多，如下：

1、models.AutoField　　 自增列 = int(11)

如果没有的话，默认会生成一个名称为 id 的列，如果要显示的自定义一个自增列，必须将给列设置为主键 primary\_key=True。

2、models.CharField　　 字符串字段，必须 max\_length 参数

3、models.BooleanField　　 布尔类型=tinyint(1)，不能为空，Blank=True

4、models.ComaSeparatedIntegerField　　用逗号分割的数字=varchar继承CharField，所以必须 max\_lenght 参数

5、models.DateField　　 日期类型 date对于参数，auto\_now = True 则每次更新都会更新这个时间；auto\_now\_add 则只是第一次创建添加，之后的更新不再改变。

6、models.DateTimeField　 日期类型 datetime 同DateField的参数

7、models.Decimal　　 十进制小数类型 = decimal必须指定整数位max\_digits和小数位decimal\_places

8、models.EmailField　　 字符串类型（正则表达式邮箱） =varchar对字符串进行正则表达式

10、models.FloatField　　 浮点类型 = double

10、models.IntegerField　　 整形

11、models.BigIntegerField　　 长整形

integer\_field\_ranges = {

　　　　 'SmallIntegerField': (-32768, 32767),

　　　　 'IntegerField': (-2147483648, 2147483647),

'BigIntegerField': (-10223372036854775808, 10223372036854775807),

　　　　 'PositiveSmallIntegerField': (0, 32767),

　　　　 'PositiveIntegerField': (0, 2147483647),

　　 }

12、models.IPAddressField　　 字符串类型（ip4正则表达式）

13、models.GenericIPAddressField　　 字符串类型（ip4和ip6是可选的）参数protocol可以是：both、ipv4、ipv6，验证时，会根据设置报错

14、models.NullBooleanField　　 允许为空的布尔类型

15、models.PositiveIntegerFiel　　 正Integer

16、models.PositiveSmallIntegerField　　 正smallInteger

17、models.SlugField　　 减号、下划线、字母、数字

18、models.SmallIntegerField　　 数字数据库中的字段有：tinyint、smallint、int、bigint

19、models.TextField　　 字符串=longtext

20、models.TimeField　　 时间 HH:MM[:ss[.uuuuuu]]

21、models.URLField　　 字符串，地址正则表达式

22、models.BinaryField　　 二进制

23、models.ImageField 图片

24、models.FilePathField 文件

常用的字段类型如上，还有一些字段常用的参数我们来认识一下

1、null=True 数据库中字段是否可以为空

2、blank=True django的 Admin 中添加数据时是否可允许空值

3、primary\_key = False 主键，对AutoField设置主键后，就会代替原来的自增 id 列

4、auto\_now 和 auto\_now\_add

　　auto\_now 自动创建---无论添加或修改，都是当前操作的时间

　　auto\_now\_add 自动创建---永远是创建时的时间

5、choices

GENDER\_CHOICE = (

(u'M', u'Male'),

(u'F', u'Female'),

)

gender = models.CharField(max\_length=2,choices = GENDER\_CHOICE)

6、max\_length

7、default　　 默认值

8、verbose\_name　　 Admin中字段的显示名称

10、name|db\_column　　 数据库中的字段名称

10、unique=True　　 不允许重复

11、db\_index = True　　 数据库索引

12、editable=True　　 在Admin里是否可编辑

13、error\_messages=None　错误提示

14、auto\_created=False　　 自动创建

15、help\_text　　 在Admin中提示帮助信息

16、validators=[]

17、upload-to 上传到哪个位置,更多与image,filepath配合使用

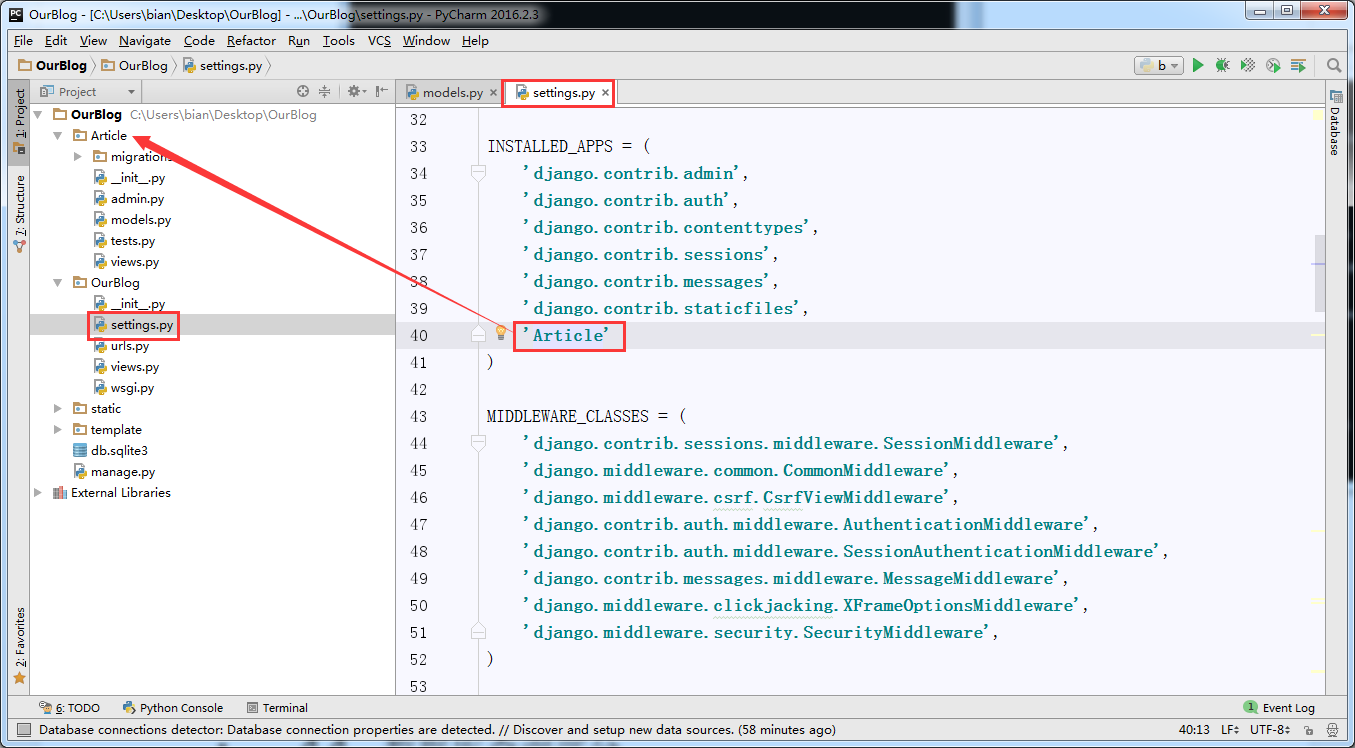
18、on\_delete 例如on\_delete = models.CASCADE, # 删除关联数据,与之关联也删除 。注意：老版本django不写没事，新版本不写会出现错误（外键关联中）

结合丰富的字段类型和对每个字段灵活的参数设置，django可以实现大部分的数据库建模，当然在必要的时候还是需要重写我们的模型类的，毕竟django的高度集成影响到了项目的灵活性，在后面的高级部分我们来探讨一下django 模型的自定义开发。

在此之前，现阶段刚刚告诉了大家如何去定义一个数据库模型，还有模型的注册，数据库的配置没有了解。先学习一下注册模型。

在django当中，与其说是注册模型，更规范的说法是注册APP功能，毕竟模型记录在功能里面。配置在settings当中，

在settings当中有配置项INSTALL\_APPS 这个配置下，填入功能名称的字符串形式。



这个时候，对数据库表的描述可以告一段落了，但是还没有对数据库进行描述，同样需要在settings当中对数据库进行描述。



Django默认配置指向sqllite数据库，支持的数据库常用的有：

postgresql、mysql、oracle、sqlite、SAP SQL Anywhere、IBM DB2、Microsoft SQL Server、Firebird、ODBC

官方优先推荐PostgreSQL关系数据库。

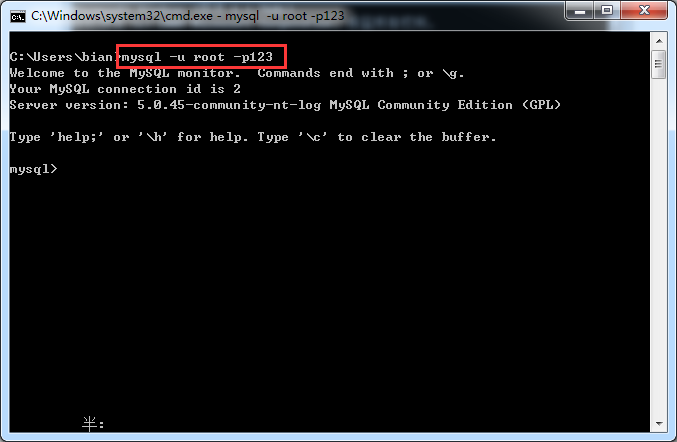
除了sqllite的默认配置之外，在这里我们列出django-mysql的配置

Django要想成功的调用mysql数据库首先保证开发环境拥有mysql数据库和python对接mysql数据库的api: MySQLdb(py2)或者pymysql(py3)模块，否则同步，调用，不会成功,

其次mysql需要创建相应名称和编码的数据库，命令如下：（在这里交给大家两种方法）

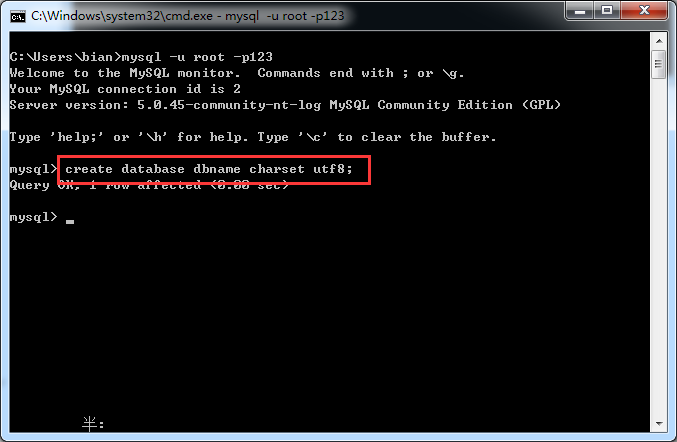
第一种：

Mysql –u root –p123 这条命令是通过命令窗口登录mysql数据库的命令，要注意的是 –u后面加空格跟着数据库的名称，-p后面不加空格跟着数据库的密码



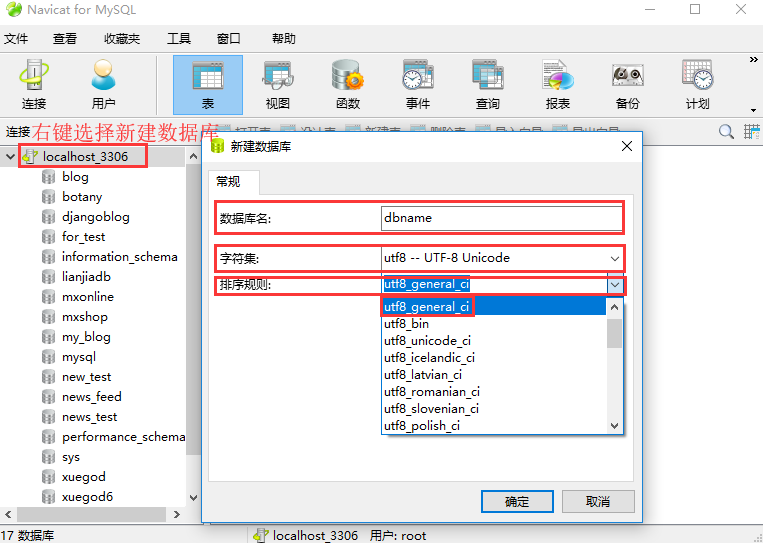
然后执行创建命令

create database dbname charset utf8;

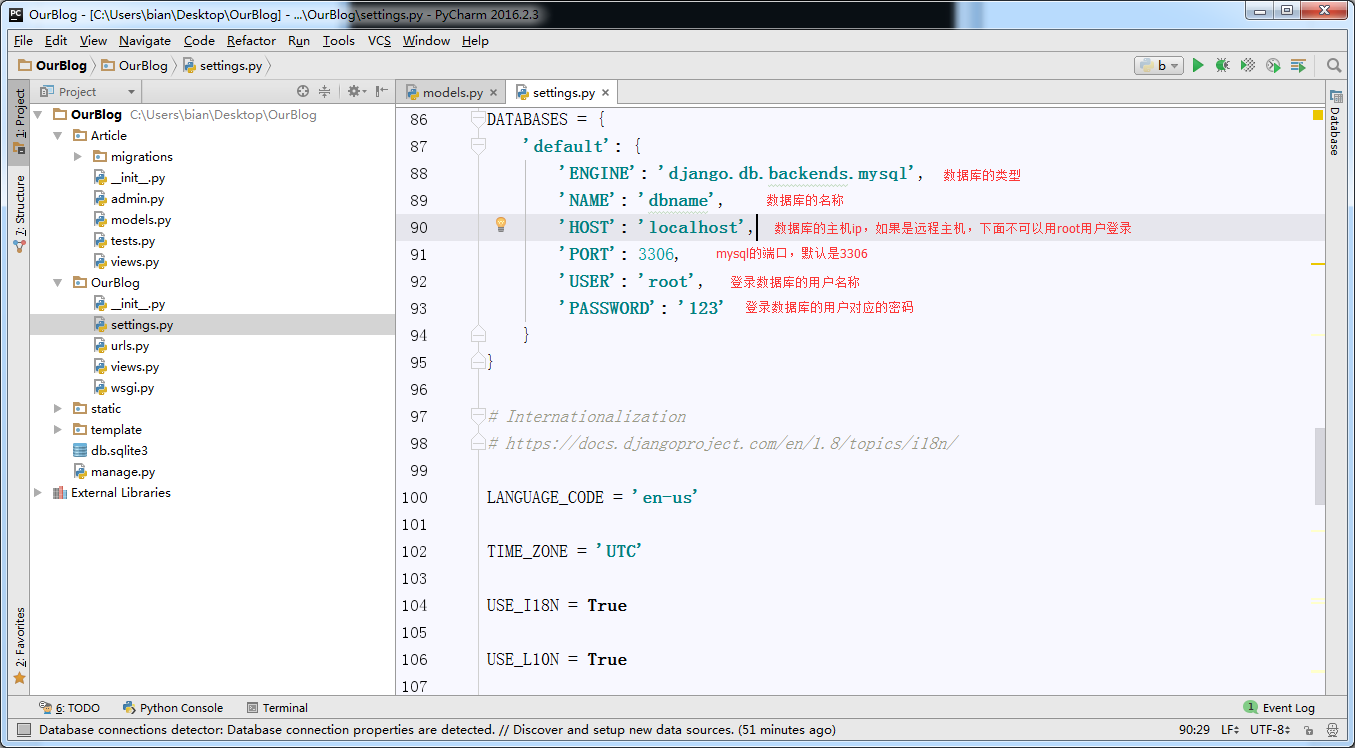


第二种：

当然我们也可以快速创建，打开自己Navicat数库图形操作界面



接着才可以在django当中配置mysql数据库



在配置完成数据库之后，我们要干的就是对数据库进行同步。

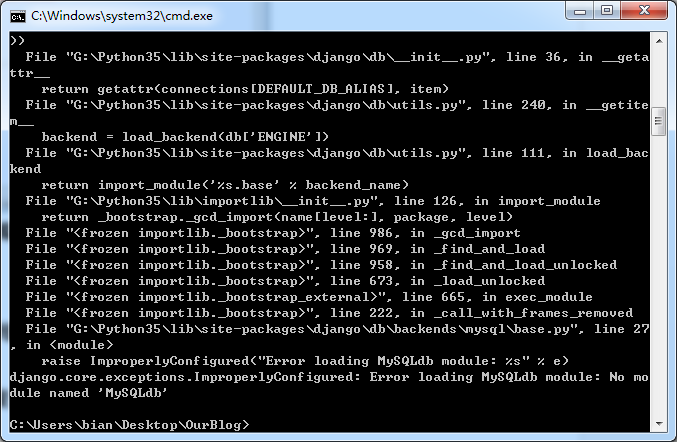
Django在数据库同步的时候分为三步：

第一步 检测配置和模型编写是否有误：

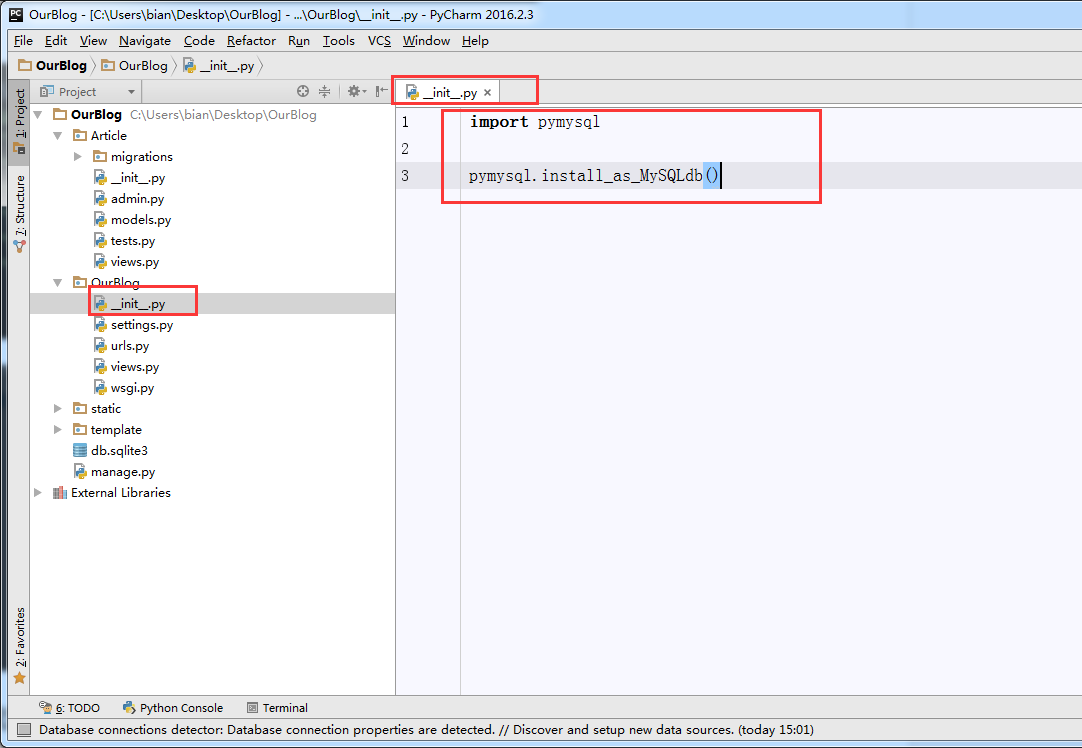
命令：python manage.py check

注意：

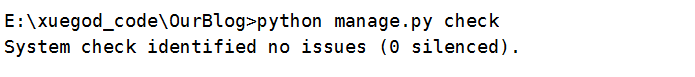
这个报错是由于python版本升级，对msyql数据库支持的模块有MySQLdb变成pymysql导致的。



需要我们做如下改动：

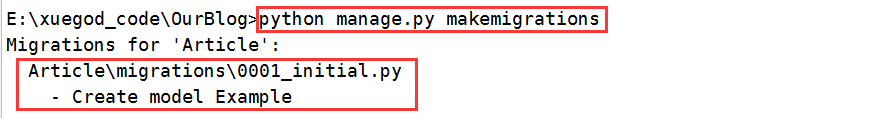


当出现这种情况，证明语法没有问题

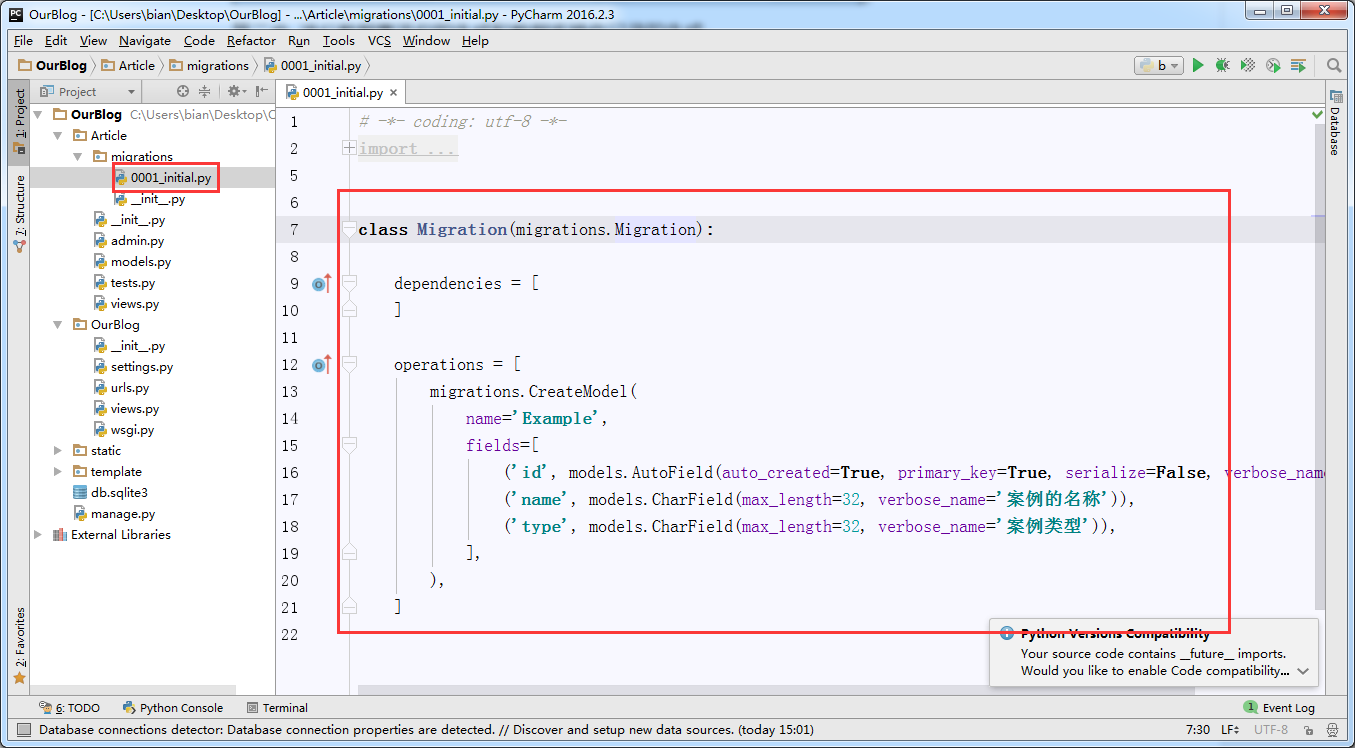


第二步 进行数据集语句的生成和数据库操作记录的生成

生成数据库语句：python manage.py makemigrations



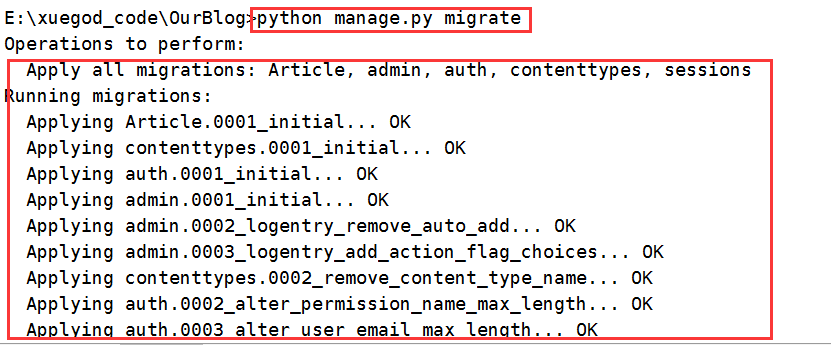
生成数据库操作日志



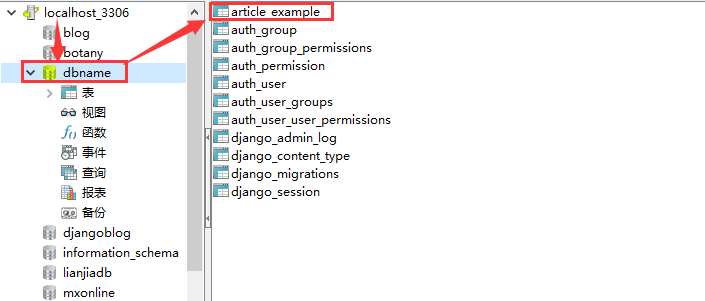
第三步 进行数据库的同步

现在我们使用的是python manage.py migrate [appname] 可指定app单独迁移

python manage.py syncdb（这个方法在python1.10版本已经停止）



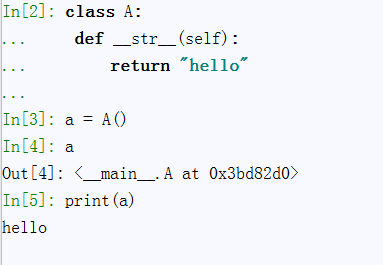
这样，就完成了功能的注册和数据表的同步，启动项目，查看效果。



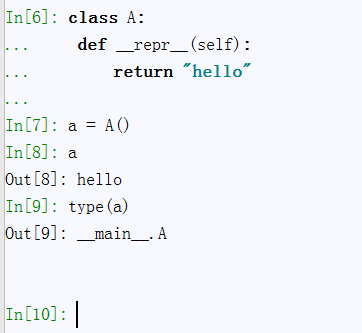
## 10.4 ORM数据库查询部分

在认识数据库查询之前，首先回忆python面向对象的两个魔术方法，

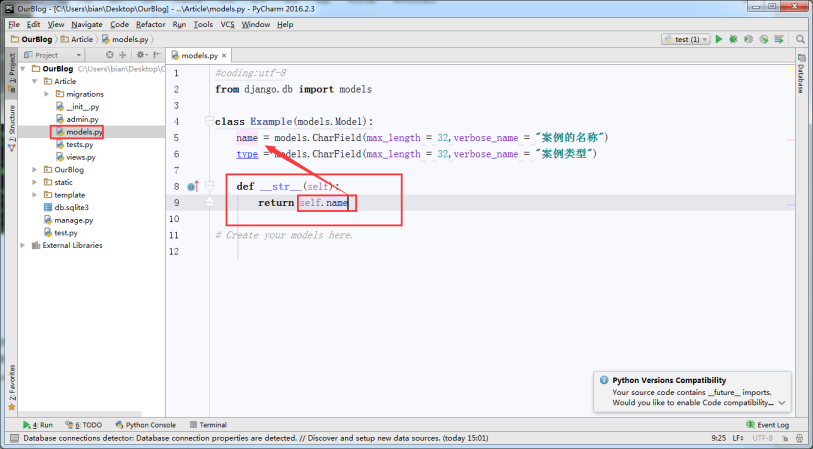
\_\_str\_\_ 当类当中设定了\_\_str\_\_之后，凡是要打印类的实例，都会返回指定的这个字符串



\_\_repr\_\_ 而repr会更加的彻底，实例之间返回名称，但是大家要注意的就是这里返回的是hello，但是a实例并没有变成字符串



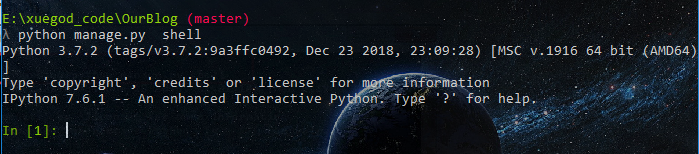
所以，在django 的models当中，开发者也可以定义\_\_str\_\_方法来让返回的实例更加明确



在上面的学习过程当中，完成了对数据模型的定义和同步，现在django项目已经连接上了数据库，紧接着就应该是数据库操作了，不用多说数据库操作分为、删、改、查4种方法，依次来认识一下orm的这四种操作的方法：

Django的数据库操作都是使用ORM的，所以不论在哪里调用数据库首先要实例化orm数据库类，为了大家对数据库操作有直观的认识，先认识一下属于django项目的shell模式

在cmd当中，切入到项目的根目录，存放manage.py的目录，然后执行python manage.py shell 获得单单数据这个项目的交互式界面。（当然，可能下面的界面有点特殊，因为这个是安装了iPython的效果，pip install ipython）

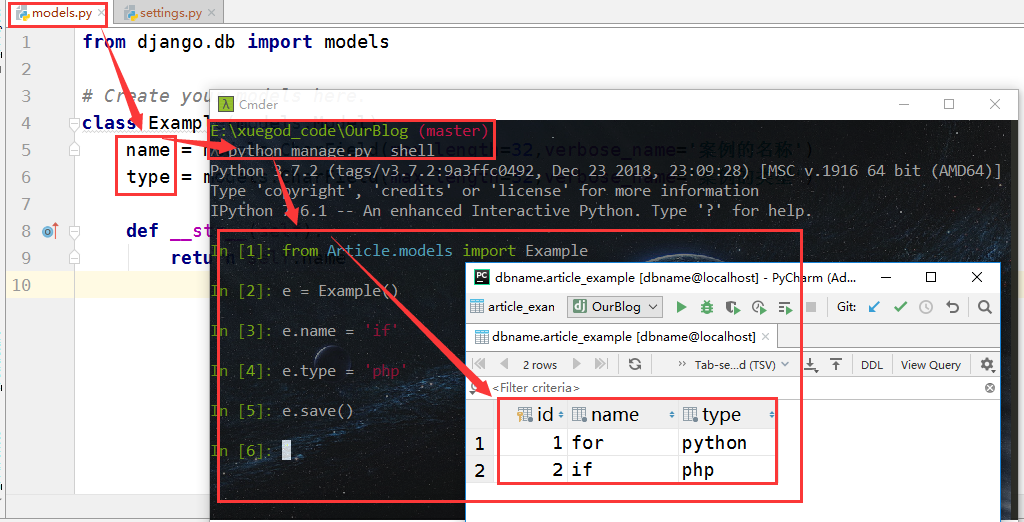


在这个界面当中开发者可以轻松的导入项目的模块进行开发，为了理解方便，先把项目的模型文件当中的代码截图出来放到上面。

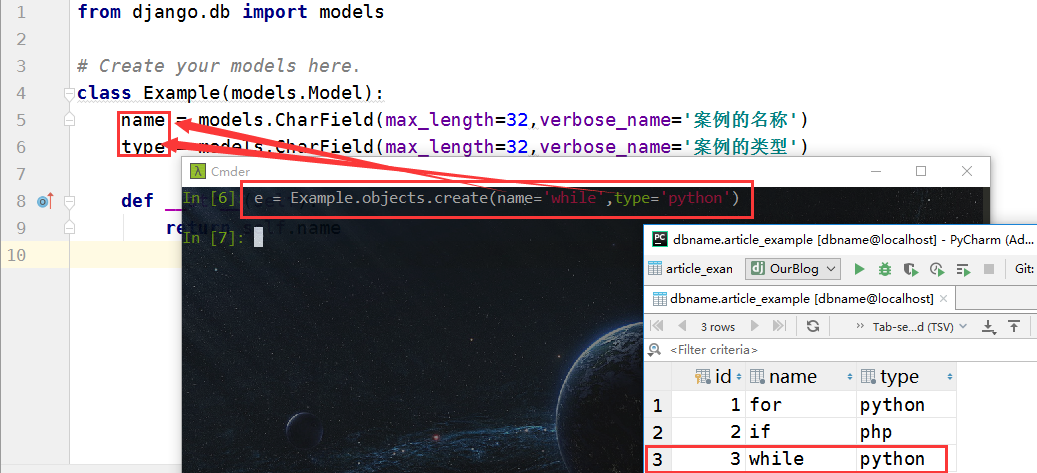


增

使用实例化的方法



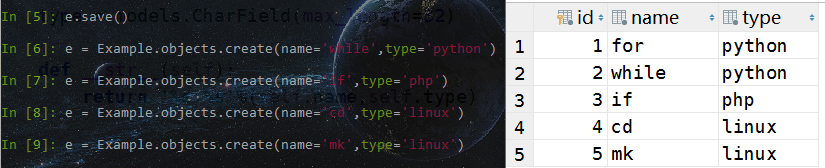
使用Model.objects.create()语句对字段进行赋值，可以实现一句话保存数据



查

由于在django项目当中，最多的就是对数据的查看并且在删除和修改数据之前都需要先捕获数据，所以先研究一下数据的查看。查看分为以下几种，为了方便查询，首先更改返回时识别的字段和在案例表里添加多条数据。

插入数据：



查所有：

Model.objects.all() 查询所有

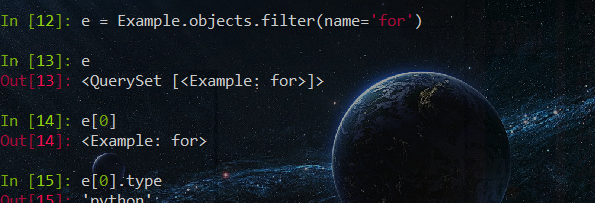


条件查：

Model.objects.filter()

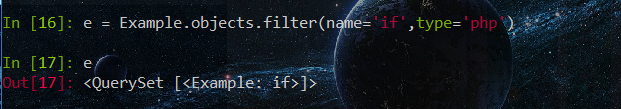
单条件查询

Filter里面指定字段等于值进行筛选

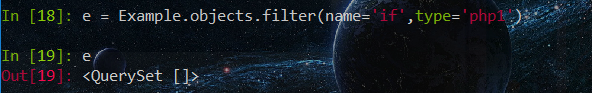


多条件查询

多条件以逗号分割查询条件，以前面的条件为主要条件

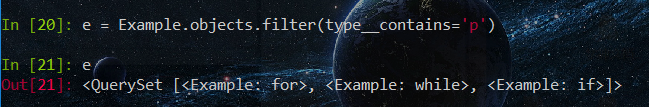


如果找不到，出现空列表



条件模糊查：

\_\_contains = “字符”按照字符模糊查询



这里要注意了，\_\_contians不是唯一的方法，这里还可以列举一些给大家看

\_\_exact 精确等于 like ‘aaa’

\_\_iexact 精确等于 忽略大小写 ilike ‘aaa’

\_\_contains 包含 like ‘%aaa%’

\_\_icontains 包含 忽略大小写 ilike ‘%aaa%’，但是对于sqlite来说，contains的作用效果等同于icontains。

\_\_gt 大于

\_\_gte 大于等于

\_\_lt 小于

\_\_lte 小于等于

\_\_in 存在于一个list范围内

\_\_startswith 以…开头

\_\_istartswith 以…开头 忽略大小写

\_\_endswith 以…结尾

\_\_iendswith 以…结尾，忽略大小写

\_\_range 在…范围内

\_\_year 日期字段的年份

\_\_month 日期字段的月份

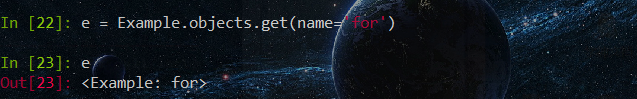
\_\_day 日期字段的日

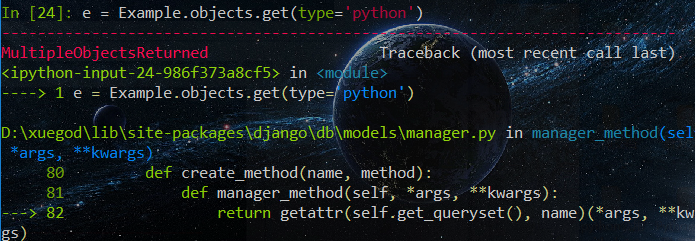
\_\_isnull=True/False

\_\_isnull=True 与 \_\_exact=None的区别

查一条：

Model.objects.get()注意在查一条的情况下，get的条件必须唯一，如果get的条件满足的数据大于一条就会报错





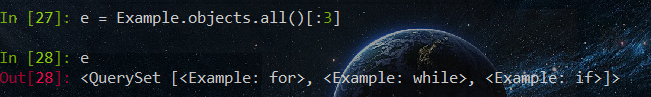
排序查：

order\_by 排序，按照指定字段排序，如果字段名字前面加符号也就是 – 就是反向排序



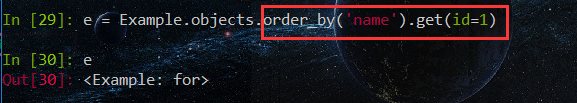
截取查：

Django ORM的截取查询实际上是按照列表的索引进行截取，看着很方便，但是，实质上是查询出了所有的数据进行截取，反而更加浪费内存,所以不是很赞成用。

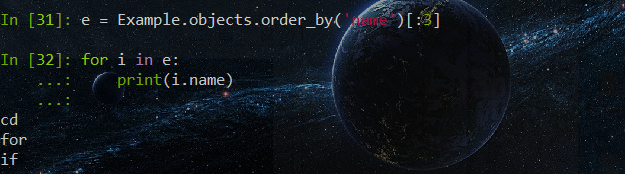


混合查：

上面所有的方法，我们都可以联合起来用的。比如：



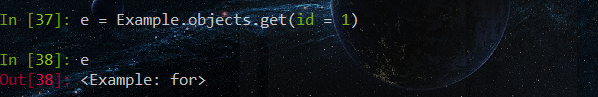
查询到数据之后，实际上返回的都是数据库模型对象，如果想获取具体的值可以用类点的方法就可以调用了。比如：

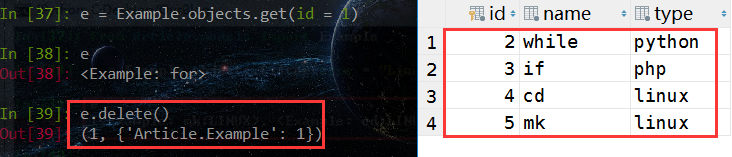


删

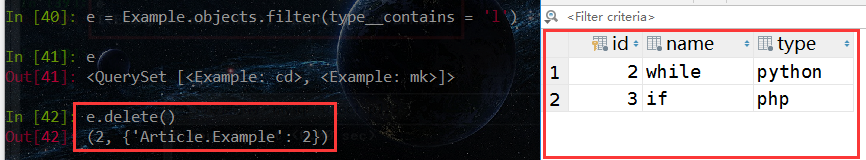
删除只有一个方法叫做 delete但是一定要慎用，因为delete方法单条和多条都可以用，在工作中我们往往会使用这个，比较常见的是逻辑删除，保证数据不会丢失

单条：





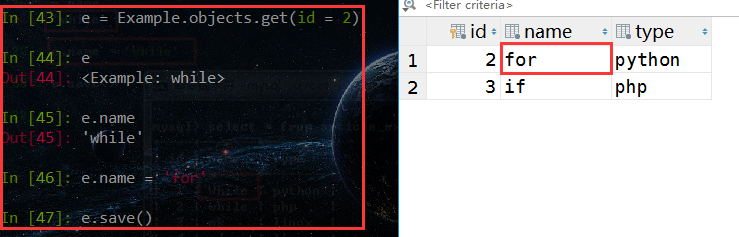
多条：



改

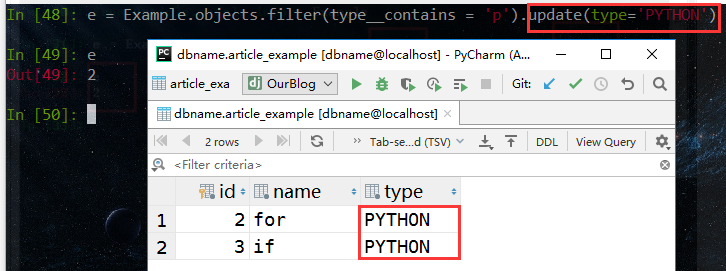
修改单个数据

修改单个数据只需要先查询到数据，然后进行修改。



修改多个数据

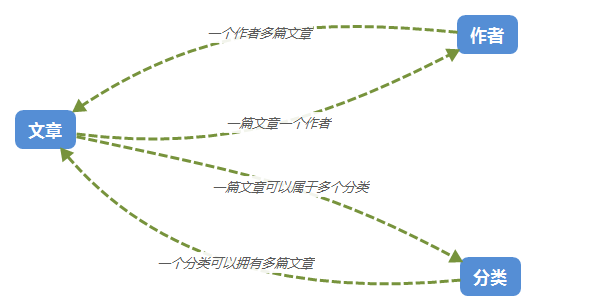
多个数据同时修改是依靠一个叫做update的方法实现的



## 10.5 ORM数据库关系部分

通过上面的学习，完成工作当中的模型搭建是没有问题的，但是刚才的课程当中也介绍过了，我们在工作当中的数据库建模除了要描述业务主体，还要描述主体之间的关系，接下来我们就开始学习django ORM部分关于数据库模型部分的知识。

首先根据我们的项目来确定一组关系：



确立关系之后，我们来搭建数据库模型

#coding:utf-8

from django.db import models

"""

一个作者多篇文章

一篇文章一个作者

一篇文章可以属于多个分类

一个分类可以拥有多篇文章

"""

class Author(models.Model):  
 gender\_choice = (  
 ("M","Male"),  
 ("F","Female"),  
 )  
 name = models.CharField(max\_length = 32,verbose\_name = "作者姓名")  
 age = models.IntegerField(verbose\_name = "作者年龄",blank = True,null = True)  
 gender = models.CharField(max\_length = 2,choices = gender\_choice,verbose\_name = "作者性别",blank = True,null = True)  
 email = models.EmailField(verbose\_name = "作者邮箱",blank = True,null = True)  
 phone = models.CharField(max\_length = 11,verbose\_name = "作者电话",blank = True,null = True)  
  
 def \_\_str\_\_(self):  
 return "作者:%s"%self.name  
  
class Classify(models.Model):  
 label = models.CharField(max\_length = 32,verbose\_name = "分类标签")  
 description = models.TextField(verbose\_name = "分类描述")  
  
 def \_\_str\_\_(self):  
 return "标签:%s" % self.label  
  
  
class Article(models.Model):  
 title = models.CharField(max\_length = 32,verbose\_name = "文章标题")  
 time = models.DateField(verbose\_name = "文章发表日期")  
 description = models.TextField(verbose\_name = "文章描述")  
 content = models.TextField(verbose\_name = "文章内容")  
  
 author = models.ForeignKey(Author,on\_delete=models.CASCADE)  
 classify = models.ManyToManyField(Classify)  
  
 def \_\_str\_\_(self):  
 return "文章:%s" % self.title

在这里，要强调以下几个点

1、\_\_str\_\_这个方法是python面向对象的魔术方法，使用之后，当有人打印类实例的时候，返回\_\_str\_\_函数的返回值



2、在Article这个类当中，我们发现了两个之前没有见过的的数据字段类型，

ForeignKey：

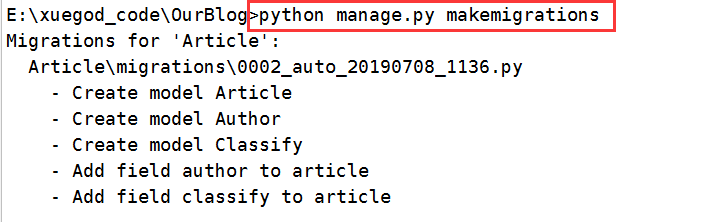
外键，多对一关系，我们用这个字段来描述，当前模型的多条数据对应于指定数据表的一条数据，比如例子当中多篇文章对应一个作者

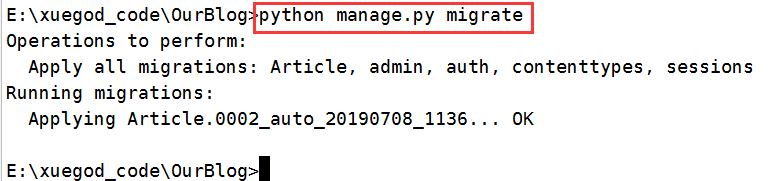
ManyToManyField

多对多关系，我们用这个字段来描述当前模型的多条数据对应指定数据表的多条数据，比如例子当中当中一篇文章有多个分类，一个分类有多篇文章

完成数据模型的搭建，按照上面介绍的步骤同步数据库，

在生成数据库描述的时候，我们发现这个时候映射的数据库表不在是三个，而是五个，其中有两个是我们的描述表





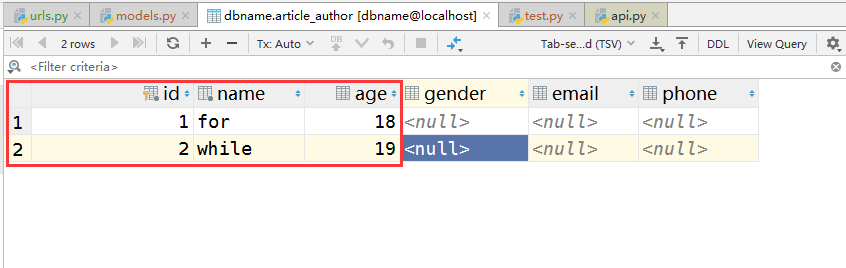
上面的讲解了数据库的建模，接下来学习关联关系的数据管理

一对多关系：

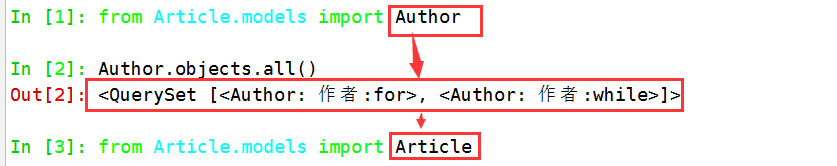
增

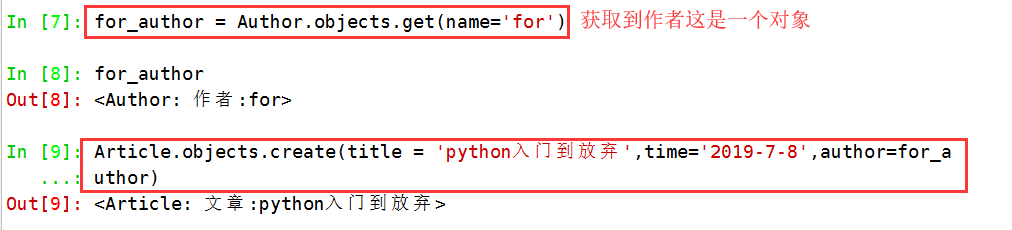
在增加数据的时候，要注意，我们一对多关系的的外键字段描述的是一个关系，那么就不可以对应一个字符串，而是一条orm数据对象。

在这里先行的给author表中添加两个作者

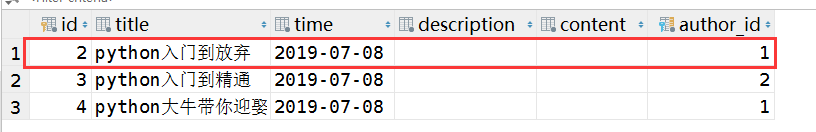


然后再在Article中添加数据，注意：外键关联的是一个对象





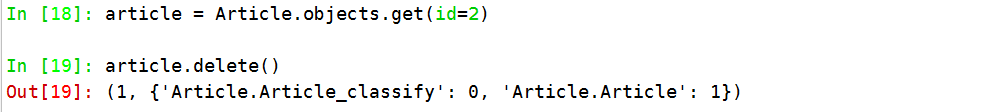
来看下表中是否有数据（插入成功）



删

同样如果我们在删除数据的时候，直接对外键表（注意这里指的是外键对应的表）进行删除的话也有点小问题。

我们可直接删除某一个文章



删除作者



所以在涉及到django外键关联的时候，我们需要考虑清楚是删除一类是删除一个，就如当前例子一个作者有多篇文章，那么我们要确定删除的是一篇文章，还是这个作者的所有文章。

改

修改部分，和之前学习的正常逻辑没有区别，我们不做过多的讲解

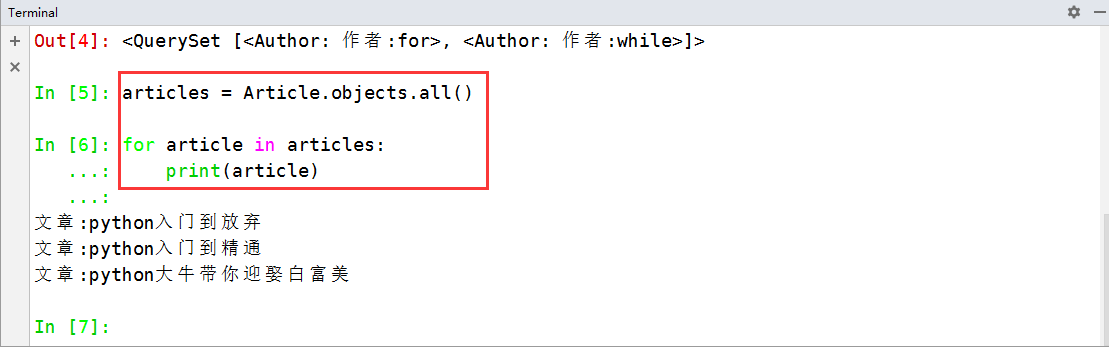
查

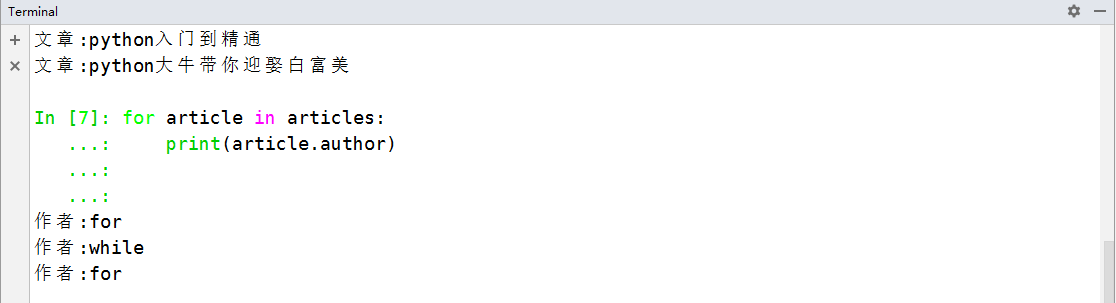
由于有关联的关系，所以我们的查询分为以下两种

多对一查询

查询每篇文章的作者







一对多查询

查询作者对应的文件

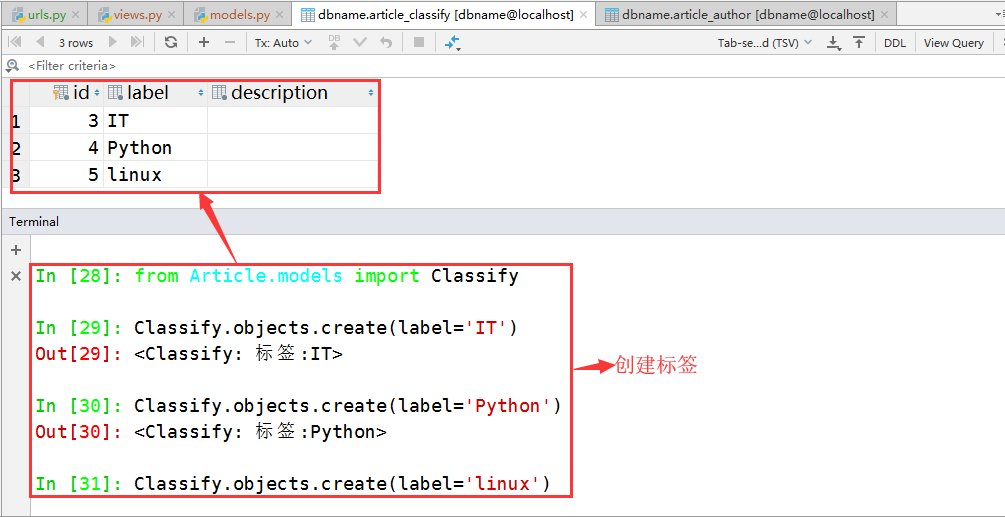


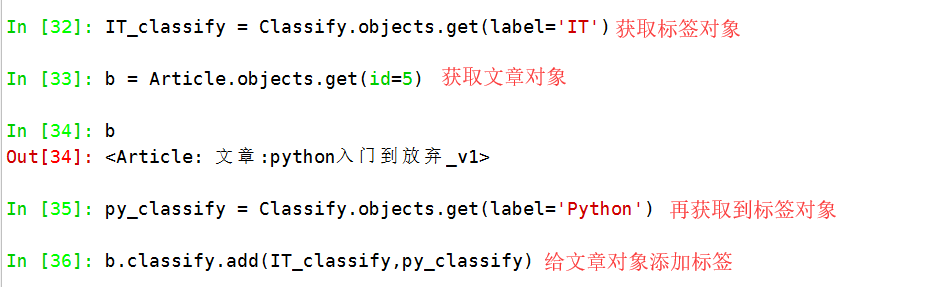
这里要注意article\_set是对应外键表名称（注意小写）加\_set形成的方法

多对多关系：

增

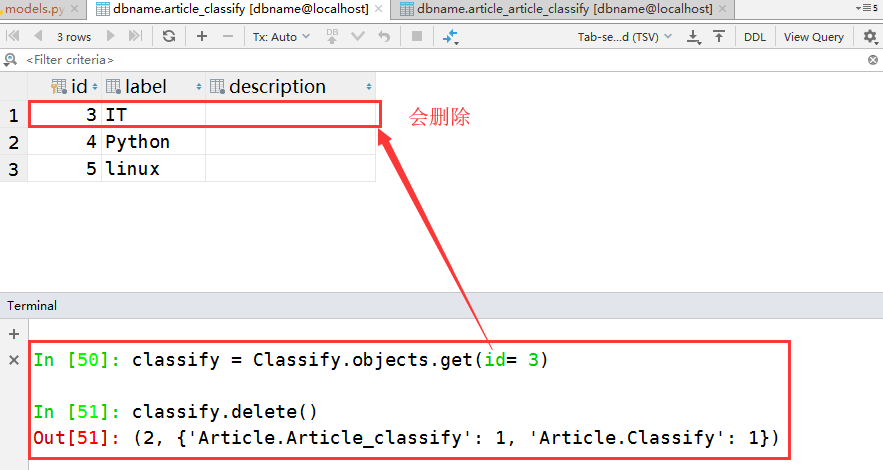
多对多的关系在增加数据的时候比较繁琐一点，我们需要先将多对多关系忽略，添加好一条数据之后，再用add方法进行多对多关联



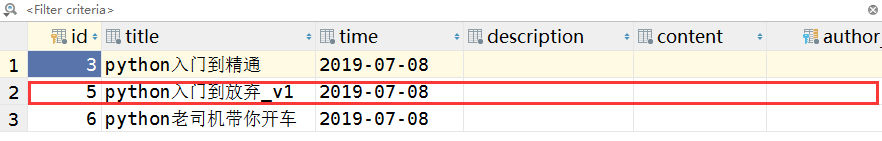


删

和多对一关系不一样的是，多对多的删除可以是正常的删除



但文章没有删除



改

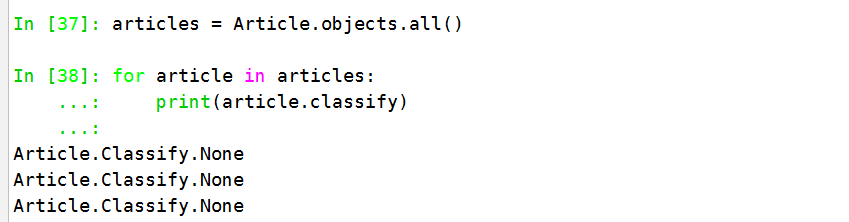
修改部分，和之前学习的正常逻辑没有区别，我们不做过多的讲解

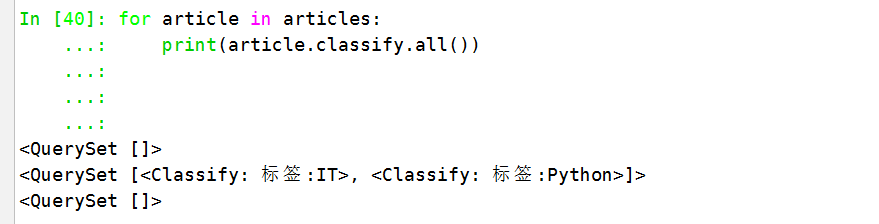
查

查询由于关系，分为两种

正向查询

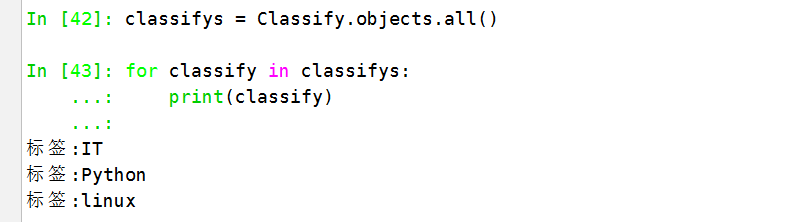
查询所有文章的类型

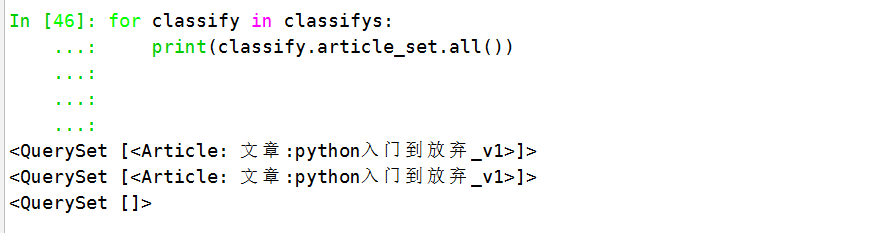




反向查询

查询指定类型的所有文章





总结：

10.1 ORM概念

10.2 Django APP功能开发

10.3 ORM数据库建模部分

10.4 ORM数据库查询部分

10.5 ORM数据库关系部分