Django数据表关联关系映射（一对一、一对多、多对多）

我们知道涉及到数据表之间的对应关系就会想到一对一、一对多、多对多，在学习 MySQL 数据库时表关系设计是需要重点掌握的知识。Django 中定义了三种关系类型的字段用来描述数据库表的关联关系：一对多（Foreignkey）、一对一（OneToOneFiled）、以及多对多（ManyToManyFiled），在本节我们对它们做简单的介绍。

1. 一对多关系类型

这种类型在数据库中体现是外键关联关系，它在和其他的 Model 建立关联同时也和自己建立关联，用来描述一对多的关系，例如一个作者可以写很多不同的书，但是这些书又只能对应这一个作者，再比如一本图书只能属于一个出版社，一个出版社可以出版很多不同种类的图书，这就是一对多的关系。Django 会自动将字段的名称添加“_id”作为列名，ForgienKey 的定义如下：

class django.db,model.ForeignKey(to,on_delete,**options)

1) 必填参数

它有两个必填参数。to，指定所关联的 Model，它的中取值可以是直接引用其他的 Model，也可以是 Model 所对应的字符串名称；on_delete，当删除关联表的数据时，Django 将根据这个参数设定的值确定应该执行什么样的 SQL 约束。

on_delete 可以理解为 MySQL 外键的级联动作，当主表执行删除操作时对子表的影响，即子表要执行的操作，Django 提供的可选值如下所示：

- CASCADE，级联删除，它是大部分 ForeignKey 的定义时选择的约束。它的表现是删除了“主”，则“子”也会被自动删除。
- PROTECT，删除被引用对象时，将会抛出 ProtectedError 异常。当主表被一个或多个子表关联时，主表被删除则会抛出异常。
- SET_NULL，设置删除对象所关联的外键字段为 null，但前提是设置了选项 null 为True，否则会抛出异常。
- SET_DEFAULT：将外键字段设置为默认值，但前提是设置了 default 选项，且指向的对象是存在的。
- SET(value)：删除被引用对象时，设置外键字段为 value。value 如果是一个可调用对象，那么就会被设置为调用后的结果。
- DO_NOTHING：不做任何处理。但是，由于数据表之间存在引用关系，删除关联数据，会造成数据库抛出异常。

2) 可选参数

除了必填参数以外，ForeignKey 还有一些常用的可选参数需要关注。如下所示：

- to_field：关联对象的字段名称。默认情况下，Django 使用关联对象的主键（大部分情况下是 id），如果需要修改成其他字段，可以设置这个参数。但是，需要注意，能够关联的字段必须有 unique=True 的约束。

- db_constraint：默认值是 True，它会在数据库中创建外键约束，维护数据完整性。通常情况下，这符合大部分场景的需求。如果数据库中存在一些历史遗留的无效数据，则可以将其设置为 False，这时就需要自己去维护关联关系的正确性了。
- related_name：这个字段设置的值用于反向查询，默认不需要设置，Django 会设置其为“小写模型名 _set”。
- related_query_name：这个名称用于反向过滤。如果设置了 related_name，那么将用它作为默认值，否则 Django 会把模型的名称作为默认值。

3) 语法格式

```
#一个A类实例对象关联多个B类实例对象
class A(model.Model):
....
class B(model.Model):
    属性 = models.ForeignKey(多对一中"一"的模型类, ...)
```

4) 实例应用

修改原来在《Django ORM进阶之项目实战》定义的代码，将出版社与图书之间修改为一对多的关系，添加如下代码：

```
from django.db import models
#新建出版社表
class PubName(models.Model):
    pubname=models.CharField('名称',max_length=255,unique=True)
#更改书籍信息表
class Book(models.Model):
    title=models.CharField(max_length=30,unique=True, verbose_name='书名')
    price=models.DecimalField(max_digits=7,decimal_places=2,verbose_name='定价')
    #添加默认价格
    def default_price(self):
        return '￥30'
    #零售价格
    retail_price=models.DecimalField(max_digits=7,decimal_places=2,verbose_name='零售价',default=default_price)
    pub=models.ForeignKey(to=PubName,on_delete=models.CASCADE ,null=True) #创建Foreign外键关联pub,以pub_id关联
    def __str__(self)
        return "title:%s pub:%s price:%s" % (self.title, self.pub, self.price)
```

此处需要注意每次更改完 models 都需要进行数据库迁移操作，依次执行以下命令即可：

```
python manager.py makemigrations
python manager.py migrate
```

插入数据创建一对多对象，如下所示：

```
#创建PubName实例化对象pub1并插入书籍信息
pub1=PubName.objects.create(pubname="清华出版社")
Book.objects.create(title="Python",price="59.00",retail_price="59.00",pub=pub1)
Book.objects.create(title="Redis",price="25.00",retail_price="25.00",pub=pub1)
Book.objects.create(title="Java",price="45.00",retail_price="45.00",pub=pub1)
#创建PubName实例化对象pub2并插入书籍信息
pub2=PubName.objects.create(pubname="c语言中文网出版")
Book.objects.create(title="Django",price="65.00",retail_price="65.00",pub=pub2)
Book.objects.create(title="Flask",price="45.00",retail_price="45.00",pub=pub2)
Book.objects.create(title="Tornado",price="35.00",retail_price="35.00",pub=pub2)
```

访问 MySQL 数据库分别查询 index_book、index_pubname 数据表（如下所示），index_pubname 数据表的 id 字段作为唯一值关联多个书籍信息，ForeignKey 外键关联键自动在 index_book 表中生成 pub_Id 字段并作为关联字段。此时 index_pubname 作为主表而 index_book 是子表，主表的 id 是子表的外键，两者之间存在外键约束 CASCADE。

```
mysql> select * from index_book;
+----+---------+--------+-------+--------------+
| id | title   | pub_id | price | retail_price |
+----+---------+--------+-------+--------------+
|  1 | Python  |      1 | 59.00 |        59.00 |
|  2 | Redis   |      1 | 25.00 |        25.00 |
|  3 | Java    |      1 | 45.00 |        45.00 |
|  4 | Django  |      2 | 65.00 |        65.00 |
|  5 | Flask   |      2 | 45.00 |        45.00 |
|  6 | Tornado |      2 | 35.00 |        35.00 |
+----+---------+--------+-------+--------------+
6 rows in set (0.01 sec)

mysql> select * from index_pubname;
+----+-----------------+
| id | pubname         |
+----+-----------------+
|  1 | c语言中文网出版   |
|  2 | 清华出版社        |
+----+-----------------+
6 rows in set (0.00 sec)
```

2. 一对一关系类型

OneToOneFiled 继承自 ForeignKey，在概念上，它类似 unique=Ture 的 ForeignKey，它与 ForeignKey 最显著的区别在于反向查询上，ForeignKey 反向查询返回的是一个对象实例列表，而 OneToOneFiled 反向查询返回的是一个对象实例。

一对关系类型的使用和场景相对其他两种关联关系要少，经常用于对已有 Model 的扩展，例如我们可以对 UserInfo 表进行扩展，添加类似用户昵称、个性签名等字段。此时就可以新建一个 Model，并定义一个字段与 UserInfo 表一对一关联。这样就实现了用户信息拓展表与 UserInfo 表一对一关联，下面会用通过实例进行说明。

1) 语法格式

```
class A(model.Model):
...
class B(model.Model):
    属性 = models.OneToOneField(A)
```

2) 实例应用

新建 index¥models.py 下添加以下代码：

```
#新建一对一关用户信息表拓展表,添加完成后执行数据库迁移同步操作
class ExtendUserinfo(models.Model):
    user=models.OneToOneField(to=UserInfo,on_delete=models.CASCADE)
```

```
    signature=models.CharField(max_length=255,verbose_name='用户签名',help_text='自建签名')
    nickname=models.CharField(max_length=255,verbose_name='昵称',help_text='自建昵称')
```

使用 Django shell 创建数据，如下所示：

```
from index.models import UserInfo,ExtendUserinfo
username=UserInfo.objects.create(username="xiaoming",password="******")
username=UserInfo.objects.create(username="xiaohong",password="*******",gender="F")
#创建一对一表关联
ExtendUserinfo.objects.create(user=username,signature="good good study,day day up",nickname="XH")
```

3) MySQL数据表显示

最后通过访问 MySQL 数据库，我们可以得到如下所示数据表，使用 user_id 进行表之间的关联：

```
mysql> select * from index_userinfo;
+----+----------+----------+--------+
| id | username | password | gender |
+----+----------+----------+--------+
|  1 | xiaoming | ******   | M      |
|  2 | xiaohong | *******  | F      |
+----+----------+----------+--------+
2 rows in set (0.00 sec)

mysql> select * from index_extenduserinfo;
+----+---------------------------+----------+---------+
| id | signature                 | nickname | user_id |
+----+---------------------------+----------+---------+
|  1 | good good study,day day up | XH       |       2 |
+----+---------------------------+----------+---------+
1 row in set (0.00 sec)
```

3. 多对多关系类型

多对多关系也是比较常见的，比如一个作者可以写很多本书，一本书也可以由很多作者一起完成，那么这时候 Author 和 Book 之间就是多对多的关系。Django 通过中间表的方式来实现 Model 之间的多对多的关系，这和 MySQL 中实现方式是一致的。这个中间表我们可以自己提供，也可以使用 Django 默认生成的中间表。

1) ManyToManyFiled定义

class django.db.models.ManyToManyFiled(to,**options)
它只有一个必填的参数即 to，与其他两个关联词在一样，用来指定与当前的 Model 关联的 Model。

2）可选参数

当然 ManyToManyFiled 还有一些重要的可选参数，下面我们对它们依次进行介绍：

- relate_name 与 ForeignKey 中的相同都用于反向查询。

- db_table 用于指定中间表的名称，如果没有提供，Django 会使用多对多字段的名称和包含这张表的 Model 的名称组合起来构成中间表的名称，当然也会包含 index 前缀。
- through 用于指定中间表，这个参数不需要设置，Django会自动生成隐式的 through Model。由于 Django可以生成自身默认的中间表，该参数可以让用户自己去控制表之间的关联关系或者增加一些额外的信息。

3) 语法格式

```
class Author(models.Model):
  ...
class Book(models.Model):
    ...
    authors = models.ManyToManyField(Author)
```

4) 多对多中间表

创建 Author 与 Book 之间多对多关联关系，在 Author Model 中添加如下代码：

```
books=models.ManyToManyField(to="Book") #创建多对多映射关系
```

然后再执行数据库迁移命令，我们可以执行以下命令可以查看 Django 执行 sql 语句：

python manage.py sqlmigrate index 0007_author_books

sql 语句如下所示：

```
CREATE TABLE `index_author_books` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `author_id` integer NOT NULL, `book_id` integer NOT NULL);
ALTER TABLE `index_author_books` ADD CONSTRAINT `index_author_books_author_id_2bfd143c_fk_index_author_id` FOREIGN KEY (`author_id`) REFERENCES `index_author` (`id`);
ALTER TABLE `index_author_books` ADD CONSTRAINT `index_author_books_book_id_1c280bc9_fk_index_book_id` FOREIGN KEY (`book_id`) REFERENCES `index_book` (`id`);
ALTER TABLE `index_author_books` ADD CONSTRAINT `index_author_books_author_id_book_id_b0dd3503_uniq` UNIQUE (`author_id`, `book_id`);
```

由 sql 语句可以看出，Django 默认隐式的创建了 index_author_books 表（Django 的命名规范），即维护关联关系的中间表。这个表有三个字段分别是主键 id，与index_author 表关联的 author_id，以及 index_book 表关联的 book_id。同时为这两个关联 id 创建了外键约束（FORGIEN KEY）,最后还为 index_author_books 表创建了唯一性约束 author_id 和 book_id。

上面介绍了 Django 自身默认创建中间表的过程，当然我们也可以自己建立中间表，然后通过 Author 表中 books 字段的 through 参数指向这张中间表。如果大家对于 MySQL 自建中间表或其它知识不熟悉，推荐学习本网站的《MySQL教程》来查漏补缺。

5) 实例应用

插入作者信息数据，如下所示：

```
author1=Author.objects.create(name="Luncy",email="123456@qq.com")
author2=Author.objects.create(name="Tom",email="456789@163.com")
```

因为书籍信息之前已经准备完毕，所以下面我们开始创建多对多映射关系，我们在 Django shell 进行如下操作：

```
author1.books.add(Book.objects.get(id="1"))
author1.books.add(Book.objects.get(id="2"))
author1.books.add(Book.objects.get(id="3"))
author2.books.add(Book.objects.get(id="1"))
author2.books.add(Book.objects.get(id="4"))
author2.books.add(Book.objects.get(id="5"))
```

```
author2.books.add(Book.objects.get(id="3"))
author2.books.add(Book.objects.get(id="6"))
author1.books.add(Book.objects.get(id="6"))
```

多对多关系在中间表插入数据需要使用 add() 方法，books 是对应的多对多字段。通过以上代码就完成多对多关系的创建，最后在 MySQL 中查看多对多相关联的三张数据表，如下所示：

#书籍信息表（index_book）

id	title	pub_id	price	retail_price
1	Python	8	59.00	59.00
2	Redis	8	25.00	25.00
3	Java	8	45.00	45.00
4	Django	9	65.00	65.00
5	Flask	9	45.00	45.00
6	Tornado	9	35.00	35.00

#作家信息表（index_author）

id	name	email
1	Luncy	123456@qq.com
2	Tom	456789@163.com

#中间表（index_author_books）

id	author_id	book_id
4	1	1
7	1	3
5	1	4
6	1	5
8	1	6
1	2	1
2	2	2
3	2	3
9	2	6

本节用了较长的篇幅给大家讲解了 Django 中数据表的关联关系，它和 MySQL 的思想是一致的，只是 Django 提供了自己的一套方法，所以我们也要学会使用它，在后续章节我们将基于此节的内容介绍 Django QuerySet API 即与数据库接口相关的表查询、更新、删除操作。