《Django Web框架教学笔记》

目录

查询数据

- 数据库的查询需要使用管理器对象进行
- 通过 MyModel.objects 管理器方法调用查询接口

方法	说明
all()	查询全部记录,返回QuerySet查询对象
get()	查询符合条件的单一记录
filter()	查询符合条件的多条记录
exclude()	查询符合条件之外的全部记录
...	

1. all()方法
    - 方法: all()
    - 用法: MyModel.objects.all()
    - 作用: 查询MyModel实体中所有的数据
        - 等同于
            - select * from tabel
    - 返回值: QuerySet容器对象,内部存放 MyModel 实例
    - 示例:

```
from bookstore.models import Book
books = Book.objects.all()
for book in books:
    print("书名", book.title, '出版社:', book.pub)
```

2. 在模型类中定义 `def __str__(self):` 方法可以自定义默认的字符串

```
class Book(models.Model):
    title = ...
    def __str__(self):
        return "书名: %s, 出版社: %s, 定价: %s" % (self.title, self.pub,
self.price)
```

3. 查询返回指定列(字典表示)
    - 方法: values('列1', '列2')
    - 用法: MyModel.objects.values(...)
    - 作用: 查询部分列的数据并返回
        - select 列1,列2 from xxx
    - 返回值: QuerySet
        - 返回查询结果容器，容器内存字典，每个字典代表一条数据,
        - 格式为: {'列1': 值1, '列2': 值2}
    - 示例:

```
from bookstore.models import Book
books = Book.objects.values("title", "pub")
for book in books:
    print("书名", book["title"], '出版社:', book['pub'])
    print("book=", book)
```

4. 查询返回指定列（元组表示)
    - 方法:values_list('列1','列2')
    - 用法:MyModel.objects.values_list(...)
    - 作用:
        - 返回元组形式的查询结果
    - 返回值: QuerySet容器对象,内部存放 `元组`
        - 会将查询出来的数据封装到元组中,再封装到查询集合QuerySet中
    - 示例:

```
from bookstore.models import Book
books = Book.objects.values_list("title", "pub")
for book in books:
    print("书名", book[0], '出版社:', book[1])
    print("book=", book)  # ('Python', '清华大学出版社')...
```

5. 排序查询
    - 方法:order_by
    - 用法:MyModel.objects.order_by('-列','列')
    - 作用:
        - 与all()方法不同，它会用SQL 语句的ORDER BY 子句对查询结果进行根据某个字段选择性的进行排序
    - 说明:
    - 默认是按照升序排序,降序排序则需要在列前增加'-'表示
    - 示例:

```
from bookstore.models import Book
books = Book.objects.order_by("price")
for book in books:
print("书名:", book.title, '定价:', book.price)
```

6. 条件查询 - filter
    - 方法: filter(条件)
    - 语法:

```
MyModel.objects.filter(属性1=值1, 属性2=值2)
```

    - 返回值:
        - QuerySet容器对象,内部存放 MyModel 实例
    - 说明:
        - 当多个属性在一起时为"与"关系，即当`Books.objects.filter(price=20, pub="清华大学出版社")`返回定价为20`且`出版社为"清华大学出版社"的全部图书
    - 示例:

```
# 查询书中出版社为"清华大学出版社"的图书
from bookstore.models import Book
books = Book.objects.filter(pub="清华大学出版社")
for book in books:
    print("书名:", book.title)

# 查询Author实体中name为王老师并且age是28岁的
    authors=Author.objects.filter(name='王老师',age=28)
```

7. 条件查询 - exclude
    - 方法: exclude(条件)
    - 语法:
        - MyModel.objects.exclude(条件)
    - 作用:
        - 返回不包含此`条件`的 全部的数据集
    - 示例:
        - 查询`清华大学出版社，定价等于50`以外的全部图书

```
books = Book.objects.exclude(pub="清华大学出版社", price=50)
for book in books:
    print(book)
```

8. 条件查询 - get
    - 方法: get(条件)
    - 语法:
        - MyModel.objects.get(条件)
    - 作用:
        - 返回满足条件的唯一一条数据
    - 说明:

        - 该方法只能返回一条数据
        - 查询结果多余一条数据则抛出,Model.MultipleObjectsReturned异常
        - 查询结果如果没有数据则抛出Model.DoesNotExist异常
    - 示例:

```
from bookstore.models import Book
book = Book.objects.get(id=1)
print(book.title)
```

查询谓词

- 每一个查询谓词是一个独立的查询功能

1. `__exact` : 等值匹配

```
Author.objects.filter(id__exact=1)
# 等同于select * from author where id = 1
```

2. `__contains` : 包含指定值

```
Author.objects.filter(name__contains='w')
# 等同于 select * from author where name like '%w%'
```

3. `__startswith` : 以 XXX 开始
4. `__endswith` : 以 XXX 结束
5. `__gt` : 大于指定值

```
Author.objects.filer(age__gt=50)
# 等同于 select * from author where age > 50
```

6. `__gte` : 大于等于
7. `__lt` : 小于
8. `__lte` : 小于等于
9. `__in` : 查找数据是否在指定范围内
    - 示例

```
Author.objects.filter(country__in=['中国','日本','韩国'])
# 等同于 select * from author where country in ('中国','日本','韩国')
```

10. `__range` : 查找数据是否在指定的区间范围内

```
# 查找年龄在某一区间内的所有作者
Author.objects.filter(age__range=(35,50))
# 等同于 SELECT ... WHERE Author BETWEEN 35 and 50;
```

11. 详细内容参见: https://docs.djangoproject.com/en/2.2/ref/models/querysets/#field-lookups

- 示例

```
MyModel.objects.filter(id__gt=4)
# 等同于 SELECT ... WHERE id > 4;
```

- 练习:
  1. 查询Book表中price大于等于50的信息
  2. 查询Author表中姓王的人的信息
  3. 查询Author表中Email中包含"w "的人的信息

修改数据

1. 修改单个实体的某些字段值的步骤:
   1. 查
      - 通过 get() 得到要修改的实体对象
   2. 改
      - 通过 对象.属性 的方式修改数据
   3. 保存
      - 通过 对象.save() 保存数据
   - 如:

```
from bookstore.models import Book
abook = Book.objects.get(id=10)
abook.market_price = "10.5"
abook.save()
```

2. 通过 QuerySet 批量修改 对应的全部字段
   - 直接调用QuerySet的update(属性=值) 实现批量修改
   - 返回值：更新数据的数量
   - 如:

```
# 将id大于3的所有图书价格定为0元
books = Book.objects.filter(id__gt=3)
books.update(price=0)
# 将所有书的零售价定为100元
books = Book.objects.all()
books.update(market_price=100)
```

删除数据

- 删除记录是指删除数据库中的一条或多条记录
- 删除单个MyModel对象或删除一个查询结果集(QuerySet)中的全部对象都是调用 delete()方法

1. 删除单个对象
   - 步骤
     1. 查找查询结果对应的一个数据对象
     2. 调用这个数据对象的delete()方法实现删除
   - 示例:

```
try:
    auth = Author.objects.get(id=1)
    auth.delete()
except:
    print(删除失败)
```

2. 删除查询结果集
    - 步骤
        1. 查找查询结果集中满足条件的全部QuerySet查询集合对象
        2. 调用查询集合对象的delete()方法实现删除
    - 示例:

```
# 删除全部作者中，年龄大于65的全部信息
auths = Author.objects.filter(age__gt=65)
auths.delete()
```

聚合查询

- 聚合查询是指对一个数据表中的一个字段的数据进行部分或全部进行统计查询,查bookstore_book数据表中的全部书的平均价格，查询所有书的总个数等,都要使用聚合查询

1. 不带分组聚合
    - 不带分组的聚合查询是指导将全部数据进行集中统计查询
    - 聚合函数【需要导入】:
        - 导入方法: `from django.db.models import *`
        - 聚合函数:
            - Sum, Avg, Count, Max, Min
    - 语法:
        - MyModel.objects.aggregate(结果变量名=聚合函数('列'))
    - 返回结果:
        - 由 结果变量名和值组成的字典
        - 格式为:
            - `{"结果变量名": 值}
    - 示例:

```
# 得到所有书的平均价格
from bookstore.models import Book
from django.db.models import Avg
result = Book.objects.aggregate(myAvg=Avg('price'))
print("平均价格是:", result['myAvg'])
print("result=", result)  # {"myAvg": 58.2}

# 得到数据表里有多少本书
from django.db.models import Count
result = Book.objects.aggregate(mycnt=Count('title'))
print("数据记录总个数是:", result['mycnt'])
print("result=", result)  # {"mycnt": 10}
```

2. 分组聚合

- 分组聚合是指通过计算查询结果中每一个对象所关联的对象集合，从而得出总计值(也可以是平均值或总和)，即为查询集的每一项生成聚合。
- 语法:
  - QuerySet.annotate(结果变量名=聚合函数('列'))
- 用法步骤:
  1. 通过先用查询结果MyModel.objects.values 查找查询要分组聚合的列
     - MyModel.objects.values('列1', '列2')
     - 如:

```
pub_set = Book.objects.values('pub')
print(pub_set)  # <QuerySet [{'pub': '清华大学出版社'}, {'pub': '清华大学出版社'}, {'pub_hou {'pub': '机械工业出版社'}, {'pub': '清华大学出版社'}]>
```

  2. 通过返回结果的 QuerySet.annotate 方法分组聚合得到分组结果
     - QuerySet.annotate(名=聚合函数('列'))
     - 返回 QuerySet 结果集,内部存储结果的字典
     - 如:

```
pub_count_set = pub_set.annotate(myCount=Count('pub'))
print(pub_count_set)  # <QuerySet [{'pub': '清华大学出版社', 'myCount': 7}, {'pub': '机械工业出版社', 'myCount': 3}]>
```

- 示例:
  - 得到哪儿个出版社共出版多少本书

```
def test_annotate(request):
    from django.db.models import Count
    from . import models

    # 得到所有出版社的查询集合QuerySet
    pub_set = models.Book.objects.values('pub')
    # 根据出版社查询分组，出版社和Count的分组聚合查询集合
    pub_count_set = pub_set.annotate(myCount=Count('pub'))  # 返回查询集合
    for item in pub_count_set:
        print("出版社:", item['pub'], "图书有：", item['myCount'])
    return HttpResponse('请查看服务器端控制台获取结果')
```

F对象

- 一个F对象代表数据库中某条记录的字段的信息

1. 作用:
   - 通常是对数据库中的字段值在不获取的情况下进行操作
   - 用于类属性(字段)之间的比较。
2. 用法
   - F对象在数据包 django.db.models 中，使用时需要先导入

        - `from django.db.models import F`

3. 语法:

```
from django.db.models import F
F('列名')
```

4. 说明:

    - 一个 F() 对象代表了一个model的字段的值
    - F对象通常是对数据库中的字段值在不加载到内存中的情况下直接在数据库服务器端进行操作

5. 示例1

    - 更新Book实例中所有的零售价涨10元

```
Book.objects.all().update(market_price=F('market_price')+10)
'UPDATE `bookstore_book` SET `market_price` =
(`bookstore_book`.`market_price` + 10)
# 以上做法好于如下代码
books = Book.objects.all()
for book in books:
    book.market_price=book.marget_price+10
    book.save()
```

6. 示例2

    - 对数据库中两个字段的值进行比较，列出哪儿些书的零售价高于定价?

```
from django.db.models import F
from bookstore.models import Book
books = Book.objects.filter(market_price__gt=F('price'))
'SELECT * FROM `bookstore_book` WHERE `bookstore_book`.`market_price` >
(`bookstore_book`.`price`)
for book in books:
    print(book.title, '定价:', book.price, '现价:', book.market_price)
```

Q对象

- 当在获取查询结果集 使用复杂的逻辑或 `|` 、 逻辑非 `~` 等操作时可以借助于 Q对象进行操作
- 如: 想找出定价低于20元 或 清华大学出版社的全部书，可以写成

```
Book.objects.filter(Q(price__lt=20)|Q(pub="清华大学出版社"))
```

- Q对象在 数据包 django.db.models 中。需要先导入再使用
    - `from django.db.models import Q`

1. 作用

    - 在条件中用来实现除 and(&) 以外的 or(|) 或 not(~) 操作

2. 运算符:

    - & 与操作
    - | 或操作
    - ~ 非操作

3. 语法

```
from django.db.models import Q
Q(条件1)|Q(条件2)  # 条件1成立或条件2成立
Q(条件1)&Q(条件2)  # 条件1和条件2同时成立
Q(条件1)&~Q(条件2)  # 条件1成立且条件2不成立
...
```

4. 示例

```
from django.db.models import Q
# 查找清华大学出版社的书或价格低于50的书
Book.objects.filter(Q(market_price__lt=50) | Q(pub_house='清华大学出版社'))
# 查找不是机械工业出版社的书且价格低于50的书
Book.objects.filter(Q(market_price__lt=50) & ~Q(pub_house='机械工业出版社'))
```

原生的数据库操作方法

- 使用MyModel.objects.raw()进行 数据库查询操作查询
  - 在django中，可以使用模型管理器的raw方法来执行select语句进行数据查询
  1. 语法:
     - `MyModel.objects.raw(sql语句，[拼接参数])`
  2. 用法
     - `MyModel.objects.raw('sql语句', [拼接参数])`
  3. 返回值:
     - RawQuerySet 集合对象 【只支持基础操作，比如循环】
  4. 示例

```
books = Book.objects.raw('select * from bookstore_book')
for book in books:
    print(book)

#sql注入问题
s1 = Book.objects.raw('select * from bookstore_book where id=%s'%('1 or 1=1'))

s2 = Book.objects.raw('select * from bookstore_book where id=%s',['1 or 1=1'])
```

- 使用django中的游标cursor对数据库进行 增删改查 操作
  - 在Django中跨国模型类直接操作数据库
  - 使用步骤:
    1. 导入cursor所在的包
       - Django中的游标cursor定义在 django.db.connection包中，使用前需要先导入
       - 如:
         - `from django.db import connection`
    2. 用创建cursor类的构造函数创建cursor对象，再使用cursor对象,为保证在出现异常时能释放cursor资源,通常使用with语句进行创建操作
       - 如:

```
from django.db import connection
with connection.cursor() as cur:
    cur.execute('执行SQL语句', '拼接参数')
```

- 示例

```
# 用SQL语句将id 为 10的 书的出版社改为 "XXX出版社"
from django.db import connection
with connection.cursor() as cur:
cur.execute('update bookstore_book set pub_house="XXX出版社" where
id=10;')

with connection.cursor() as cur:
    # 删除 id为1的一条记录
    cur.execute('delete from bookstore_book where id=10;')
```