Django查询

和DML一样，增删改不难，最复杂的就是查询。

查询集

如果查的是一批数据，那么返回的是一个结果的集合

- 它是django.db.models.query.QuerySet的实例
- 可迭代对象

1、惰性求值

创建查询集不会带来任何对数据库的访问，直到调用方法使用数据时，才会访问数据库。

在迭代、序列化、if语句、切片、len()、repr()、bool()中都会立即求值。

参考 https://docs.djangoproject.com/en/3.2/ref/models/querysets/#when-querysets-are-evaluated

2、缓存

参考 https://docs.djangoproject.com/en/3.2/topics/db/queries/#caching-and-querysets

每一个查询集都包含一个缓存，来最小化对数据库的访问。

新建查询集，缓存为空。首次对查询集求值时，会发生数据库查询，Django会把查询的结果存在这个缓存中，并返回请求的结果，接下来对查询集求值将使用缓存的结果。

通过日志，观察下面的2个例子是要看真正生成的语句了

```
from employee.models import Employee

emps = Employee.objects.all() # 结果集，本句不发起查询
print(type(emps)) # QuerySet查询集
print(1, emps)
print(2, emps)
print(3, emps[0])
print(4, emps[0])
print(emps._result_cache) # None
# 上面需要查询4次数据库
```

```
from employee.models import Employee

emps = Employee.objects.all() # 结果集，本句不发起查询
print(type(emps)) # QuerySet查询集
print(list(emps)) # print(*emps) 先遍历一遍，缓存住
print(1, emps)
print(2, emps)
print(3, emps[0])
print(4, emps[:])
print(emps._result_cache) # 结果集列表
```

限制查询集（切片）

分页功能实现，使用限制查询集。

查询集对象可以直接使用索引下标的方式（不支持负索引），相当于SQL语句中的limit和offset子句。

注意：使用切片返回的新的结果集，依然是惰性求值，不会立即查询。但是使用了切片步长，会立即查询

```
qs = Employee.objects.all()[10:15]
# LIMIT 5 OFFSET 10
qs = Employee.objects.all()[20:30]
# LIMIT 10 OFFSET 20
```

注：在使用print函数打印结果集的时候，看到SQL语句有自动添加的LIMIT 21，这是怕打印的太长了。使用for循环迭代就没了。

结果集方法

名称	返回值类型	说明
all()	QuerySet	
filter()	QuerySet	过滤，返回满足条件的数据
exclude()	QuerySet	排除，排除满足条件的数据
order_by()	QuerySet	排序，注意参数是字符串
values()	QuerySet	返回集合内的元素是字典，字典内是字段和值的键值对

返回值如果是QuerySet类型，可以链式调用。

`filter(k1=v1).filter(k2=v2)`等价于`filter(k1=v1, k2=v2)`

`filter(pk=10)`这里pk指的就是主键， 不用关心主键字段名，当然也可以使用使用主键名`filter(emp_no=10)`

```
mgr = Employee.objects
print(mgr.all())
print(mgr.values())
print(mgr.filter(pk=10010).values())
print(mgr.exclude(emp_no=10001))
print(mgr.exclude(emp_no=10002).order_by('emp_no'))
print(mgr.exclude(emp_no=10002).order_by('-pk'))
print(mgr.exclude(emp_no=10002).order_by('gender', '-pk').values()) # 返回依然是QuerySet
```

`filter(pk=10010)`这是关键字传参，所以不能写成`filter(pk!=10010)`或`filter(pk>=10010)`

返回单个值的方法

名称	说明
get()	严格返回满足条件的单个对象 如果未能返回对象则抛出DoesNotExist异常；如果能返回多条，抛出MultipleObjectsReturned异常
count()	返回当前查询的总条数
first()	返回第一个对象
last()	返回最后一个对象
exist()	判断查询集中是否有数据，如果有则返回True

```
mgr = Employee.objects
print(mgr.filter(pk=10010).get())
print(mgr.get(pk=10001))
#print(mgr.exclude(pk=10010).get()) # get严格一个

print(mgr.first()) # limit 1
print(mgr.exclude(pk=10010).last()) # desc, limit 1
print(mgr.filter(pk=10010, gender=1).first()) # AND, 找不到返回None
print(mgr.count())
print(mgr.exclude(pk=10010).count())
```


字段查询（Field Lookup）表达式

参考 https://docs.djangoproject.com/en/3.2/ref/models/querysets/#field-lookups

字段查询表达式可以作为filter()、exclude()、get()的参数，实现where子句。

语法：`属性名称__比较运算符=值`。注意：属性名和运算符之间使用双下划线

名称	举例	说明
exact	filter(isdeleted=False) filter(isdeleted__exact=False)	严格等于，可省略不写
contains	exclude(title__contains='天')	是否包含，大小写敏感，等价于 `like binary '%天%'` 模糊匹配效率很低
startswith endswith	filter(title__startswith='天')	以什么开头或结尾，大小写敏感
isnull isnotnull	filter(title__isnull=False)	是否为null
iexact icontains istartswith iendswith		i的意思是忽略大小写
in	filter(pk__in=[1,2,3,100])	是否在指定范围数据中

gt、gte
lt、lte
	`filter(id__gt=3)`
	`filter(pk__lte=6)`
	`filter(pub_date__gt=date(2000,1,1))`
		大于、小于等

year month day week_day hour minute second	filter(pub_date__year=2000)	对日期类型属性处理


```
mgr = Employee.objects
print(mgr.filter(emp_no__exact=10010)) # 就是等于，所以很少用exact
print(mgr.filter(pk__in=[10010, 10009]))
print(mgr.filter(last_name__startswith='P'))
print(mgr.exclude(pk__gt=10003))
```

Q对象

虽然Django提供传入条件的方式，但是不方便，它还提供了Q对象来解决。

Q对象是django.db.models.Q，可以使用&、|操作符来组成逻辑表达式。`~`表示not。

```
from django.db.models import Q
mgr = Employee.objects

print(mgr.filter(Q(pk__lt=10006))) # 不如直接写filter(pk__lt=10006)

# 下面几句一样
```

```
print(mgr.filter(pk__gt=10003).filter(pk__lt=10006)) # 与
print(mgr.filter(pk__gt=10003, pk__lt=10006)) # 与
print(mgr.filter(Q(pk__gt=10003), Q(pk__lt=10006)))
print(mgr.filter(Q(pk__gt=10003) & Q(pk__lt=10006))) # 与
print(mgr.filter(pk__gt=10003) & mgr.filter(pk__lt=10006))

# 下面几句等价
print(mgr.filter(pk__in=[10003, 10006])) # in
print(mgr.filter(Q(pk=10003) | Q(pk=10006))) # 或
print(mgr.filter(pk=10003) | mgr.filter(pk=10006))

print(mgr.filter(~Q(pk__gt=10003))) # 非
```

可使用&|和Q对象来构造复杂的逻辑表达式，可以使用一个或多个Q对象。
如果混用关键字参数和Q对象，那么Q对象必须位于关键字参数的前面。

聚合、分组

aggregate() 返回字典，方便使用

```
from employee.models import Employee
from django.db.models import Q, Avg, Sum, Max, Min, Count

mgr = Employee.objects
print(mgr.filter(pk__gt=10010).count()) # 单值
print(mgr.filter(pk__gt=10010).aggregate(Count('pk'), Max('pk'))) # 字典
print(mgr.filter(pk__lte=10010).aggregate(Avg('pk')))
print(mgr.aggregate(Max('pk'), min=Min('pk'))) # 别名
```

annotate()方法用来分组聚合，返回查询集。

```
mgr = Employee.objects
print(mgr.filter(pk__gt=10010).aggregate(Count('pk'))) # 字典
s = mgr.filter(pk__gt=10010).annotate(Count('pk')) # 返回查询集，没指定分组字段，使用主键分组
for x in s:
    print(x)
    print(x.__dict__) # 里面多了一个属性pk__count
```

values()方法，放在annotate前就是指定分组字段，之后就是取结果中的字段。

```
mgr = Employee.objects
s = mgr.filter(pk__gt=10010).values('gender').annotate(Count('pk')) # 查询集
print(s)
for x in s:
    print(x) # 字典

# 运行结果如下
<QuerySet [{'gender': 2, 'pk__count': 3}, {'gender': 1, 'pk__count': 7}]>
{'gender': 2, 'pk__count': 3}
{'gender': 1, 'pk__count': 7}
```

```
mgr = Employee.objects
s =
mgr.filter(pk__gt=10010).values('gender').annotate(c=Count('pk')).order_by('
-c') # 查询集
print(s)
for x in s:
    print(x) # 字典

# 运行结果如下
<QuerySet [{'gender': 1, 'c': 7}, {'gender': 2, 'c': 3}]>
{'gender': 1, 'c': 7}
{'gender': 2, 'c': 3}
```

```
mgr = Employee.objects
s = mgr.filter(pk__gt=10010).values('gender').annotate(
    Avg('pk'), c=Count('pk')
).order_by('-c').values('pk__avg', 'c') # 查询集,但后面的values过滤了每个对象字典
的key
print(s)
for x in s:
    print(x) # 字典

# 运行结果如下
<QuerySet [{'pk__avg': 10015.5714, 'c': 7}, {'pk__avg': 10015.3333, 'c':
3}]>
{'pk__avg': 10015.5714, 'c': 7}
{'pk__avg': 10015.3333, 'c': 3}
```