



장고 차근차근 시작하기 2/E

당신의 파이썬/장고 페이스메이커가 되겠습니다. ☺

EP-13. 모델을 통한 조회 (기초)

Model Manager ([공식문서](#))

- 데이터베이스 질의 인터페이스를 제공
- 디폴트 Manager로서 **ModelCls.objects** 가 제공

```
# 생성되는 대강의 SQL 윤곽 → SELECT * FROM app_model;  
ModelCls.objects.all()
```

```
# 생성되는 대강의 SQL 윤곽 → SELECT * FROM app_model ORDER BY id DESC LIMIT 10;  
ModelCls.objects.all().order_by('-id')[:10]
```

```
# 생성되는 대강의 SQL 윤곽 → INSERT INTO app_model (title) VALUES ("New Title");  
ModelCls.objects.create(title="New Title")
```

QuerySet ([querysets](#), [queries](#))

- SQL을 생성해주는 인터페이스
- 순회가능한 객체
- Model Manager를 통해, 해당 Model에 대한 QuerySet을 획득
 - `Post.objects.all()` 코드는 "SELECT * FROM post;" ...
 - `Post.objects.create(...)` 코드는 "INSERT INTO ...;"...

QuerySet은 Chaining을 지원

- `Post.objects.all().filter(...).exclude(...).filter(...)` → QuerySet
- QuerySet은 Lazy한 특성
 - QuerySet을 만드는 동안에는 DB접근을 하지 않습니다.
 - 실제로 데이터가 필요한 시점에 접근을 합니다.
- 데이터가 필요한 시점은 언제인가?
 1. `queryset`
 2. `print(queryset)`
 3. `list(queryset)`
 4. `for instance in queryset: print(instance)`

다양한 조회요청 방법

SELECT SQL 생성

- 조건을 추가한 Queryset, 획득할 준비
 - queryset.filter(...) → queryset
 - queryset.exclude(...) → querriyset
- 특정 모델객체 **1개** 획득을 시도
 - queryset[숫자인덱스]
 - → 모델객체 혹은 예외발생 (IndexError)
 - queryset.get(...)
 - → 모델객체 혹은 예외발생 (DoesNotExist , MultipleObjectsReturned)
 - queryset.first() → 모델객체 혹은 None
 - queryset.last() → 모델객체 혹은 None

filter ↔ exclude

SELECT 쿼리에 WHERE 조건 추각

- 인자로 "필드명 = 조건값" 지정
- 1개 이상의 인자 지정 → 모두 AND 조건으로 묶임.
- OR 조건을 묶으려면, django.db.models.Q 활용

```
In [26]: Item.objects.filter(name="New Item", price=3000)
Out[26]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" WHERE ("shop_item"."name" = 'New Item' AND "shop_item"."price" = 3000) LIMIT 21
```

Execution time: 0.000154s [Database: default]

<QuerySet []>

```
In [27]: Item.objects.exclude(name="New Item", price=3000)
Out[27]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" WHERE NOT ("shop_item"."name" = 'New Item' AND "shop_item"."price" = 3000) LIMIT 21
```

Execution time: 0.000232s [Database: default]

<QuerySet [<Item: Item object (1)>, <Item: Item object (2)>, <Item: Item object (3)>, <Item: Item object (4)>]>

OR 조건 추가

```
In [28]: from django.db.models import Q
```

```
In [29]: Item.objects.filter(Q(name="New Item") & Q(price=3000))
```

```
Out[29]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" WHERE ("shop_item"."name" = 'New Item' AND "shop_item"."price" = 3000) LIMIT 21
```

Execution time: 0.000188s [Database: default]

<QuerySet []>

```
In [30]: Item.objects.filter(Q(name="New Item") | Q(price=3000))
```

```
Out[30]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" WHERE ("shop_item"."name" = 'New Item' OR "shop_item"."price" = 3000) LIMIT 21
```

Execution time: 0.000221s [Database: default]

<QuerySet []>

필드 타입별 다양한 조건 매칭

주의) 데이터베이스에 따라 생성되는 SQL이 다릅니다.

• 숫자/날짜/시간 필드

- 필드명_lt = 조건값 → 필드명 < 조건값
- 필드명_lte = 조건값 → 필드명 <= 조건값
- 필드명_gt = 조건값 → 필드명 > 조건값
- 필드명_gte = 조건값 → 필드명 >= 조건값

• 문자열 필드

- 필드명_startswith = 조건값 → 필드명 LIKE "조건값%"
- 필드명_endswith = 조건값 → 필드명 LIKE "%조건값"
- 필드명_contains = 조건값 → 필드명 LIKE "%조건값%"
- 필드명_istartswith = 조건값 → 필드명 ILIKE "조건값%"
- 필드명_iendswith = 조건값 → 필드명 ILIKE "%조건값"
- 필드명_icontains = 조건값 → 필드명 ILIKE "%조건값%"

• ETC

실전예제) Item 목록/간단검색 페이지

```
# shop/views.py
```

```
from django.shortcuts import render
from .models import Item
```

```
# 중략 ...
```

```
def item_list(request):
    qs = Item.objects.all()

    q = request.GET.get('q', '')
    if q:
        qs = qs.filter(name__icontains=q)

    return render(request, 'shop/item_list.html', {
        'item_list': qs,
        'q': q,
    })
```

```
# shop/views.py
# 중략 ...
```

```
urlpatterns = [
    path('', views.item_list),
]
```

```
<!-- shop/templates/shop/item_list.html -->
```

```
<form action="" method="GET">
    <input type="text" name="q" value="{{ q }}" />
    <input type="submit" value="검색" />
</form>
```

```
<hr/>
```

```
{% for item in item_list %}
    <div>
        <h3>{{ item.name }}</h3>
        <h4>가격: {{ item.price }}</h4>
        {% if item.desc %}
            <p>{{ item.desc }}</p>
        {% endif %}
    </div>
{% endfor %}
```

QuerySet에 정렬 조건 추가

정렬 조건 추가

SELECT 쿼리에 "ORDER BY" 추가

- 정렬 조건을 추가하지 않으면 일관된 순서를 보장받을 수 없음.
- DB에서 다수 필드에 대한 정렬을 지원
 - 하지만, 가급적 단일 필드로 하는 것이 성능에 이익
 - 시간순/역순 정렬이 필요할 경우, id 필드를 활용해볼 수 있음.
- 정렬 조건을 지정하는 2가지 방법
 1. (추천) 모델 클래스의 Meta 속성으로 ordering 설정 : list로 지정
 2. 모든 queryset에 order_by(...) 에 지정

정렬 지정하기 #1

```
class Item(models.Model):  
    name = models.CharField(max_length=100)  
    desc = models.TextField(blank=True)  
    price = models.PositiveIntegerField()  
    created_at = models.DateTimeField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)  
  
    class Meta:  
        ordering = ['id']
```

```
→ python manage.py shell_plus --print-sql
```

```
In [1]: Item.objects.all()  
Out[1]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" ORDER BY "shop_item"."id" ASC LIMIT 21
```

정렬 지정하기 #2

```
class Item(models.Model):  
    name = models.CharField(max_length=100)  
    desc = models.TextField(max_length=1000)  
    price = models.PositiveIntegerField()  
    created_at = models.DateTimeField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)
```

```
class Meta:  
    ordering = ['id']
```



→ python manage.py shell_plus --print-sql

```
In [1]: Item.objects.all()  
Out[1]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" LIMIT 21
```

Execution time: 0.000369s [Database: default]

<QuerySet []>

```
In [2]: Item.objects.all().order_by('id')  
Out[2]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" ORDER BY "shop_item"."id" ASC LIMIT 21
```

Execution time: 0.000258s [Database: default]

<QuerySet []>

```
In [3]: Item.objects.all().order_by('-id')  
Out[3]: SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at", "shop_item"."updated_at" FROM "shop_item" ORDER BY "shop_item"."id" DESC LIMIT 21
```

Execution time: 0.000140s [Database: default]

<QuerySet []>

QuerySet에 범위 조건 추가

슬라이싱을 통한 범위조건 추가

SELECT 쿼리에 "OFFSET/LIMIT" 추가

- str/list/tuple에서의 슬라이싱과 거의 유사하나, 역순 슬라이싱은 지원하지 않음.
 - 데이터베이스에서 지원하지 않기 때문.
- 객체[start:stop:step]
 - OFFSET → start
 - LIMIT → stop – start
 - (주의) step은 쿼리에 대응되지 않습니다. 사용을 비추천.

```
In [10]: Item.objects.all()[10:30:2]  
SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at",  
"shop_item"."updated_at" FROM "shop_item" LIMIT 20 OFFSET 10
```

역순 슬라이싱 대응하기

```
In [31]: qs = Item.objects.all().order_by('id')
```

```
In [32]: qs[-10:]
```

```
-----  
AssertionError                                Traceback (most recent call last)
```

```
<ipython-input-32-1a16fb98630c> in <module>()
```

```
----> 1 qs[-10:]
```

```
/anaconda3/lib/python3.6/site-packages/django/db/models/query.py in __getitem__(self, k)
```

```
    280         (isinstance(k, slice) and (k.start is None or k.start >= 0) and
```

```
    281         (k.stop is None or k.stop >= 0))), \
```

```
--> 282         "Negative indexing is not supported."
```

```
    283
```

```
    284         if self._result_cache is not None:
```

```
AssertionError: Negative indexing is not supported.
```

```
In [33]: reversed(qs.reverse()[:10])
```

```
SELECT "shop_item"."id", "shop_item"."name", "shop_item"."desc", "shop_item"."price", "shop_item"."created_at",  
"shop_item"."updated_at" FROM "shop_item" ORDER BY "shop_item"."id" DESC LIMIT 10
```

```
Execution time: 0.000313s [Database: default]
```


인생은 짧습니다.
파이썬/장고를 쓰세요.