Djagno 기반의 웹프로그래밍

이 소 영

yisy0703@naver.com

Web Frameworks for Python(https://wiki.python.org/moin/WebFrameworks)
장고 공식 사이트(https://www.djangoproject.com/)
장고 공식 소스 저장소(http://github.com/django/django)
장고 참조 문서(https://docs.djangoproject.com/ko/5.2/)

Agenda

Django(장고) 기반의 파이썬 웹 프로그래밍

Ch01. Django 시작하기
1. Django 란?
2. 개발 환경 구축
3. Django 구조

Ch02. Django App
1. Django Project
2. Model
3. View

Ch03. Model
1. Model 속성 및 옵션
2. Relationship
3. Migrations
4. Admin App

Ch04. Django SQL
1. Django shell
2. Manager & QuerySet
3. 조회 SQL
4. 생성/수정/삭제 SQL
5. Django-Debug-Toolbar

Ch05. Template
1. Template Loader
2. URL Dispatcher
3. Template 상속
4. Template Engines
5. Template Filter

Ch07. Django View
1. View 기본
2. View 활용

Ch06. Django Form
1. HTML form
2. CSRF
3. HttpRequest/HttpResponse
4. Django Form
5. Django Model Form
6. Form Validation

Ch08. File 관리
1. Static Files
2. Media Files

Ch09. 사용자 인증
1. Auth App
2. 회원가입 구현
3. 로그인/아웃 구현
4. Oauth 라이브러리 활용

Ch 04 Django SQL

1. Djagno Shell
2. Manager & QuerySet
3. 조회 SQL
4. 생성/수정/삭제 SQL
5. Django-Debug-Toolbar

Chapter 04. Django SQL

Python shell

- IPython(http://ipython.org/)
- BPython(https://bpython-interpreter.org/)
- django-extensions(https://django-extensions.readthedocs.io)

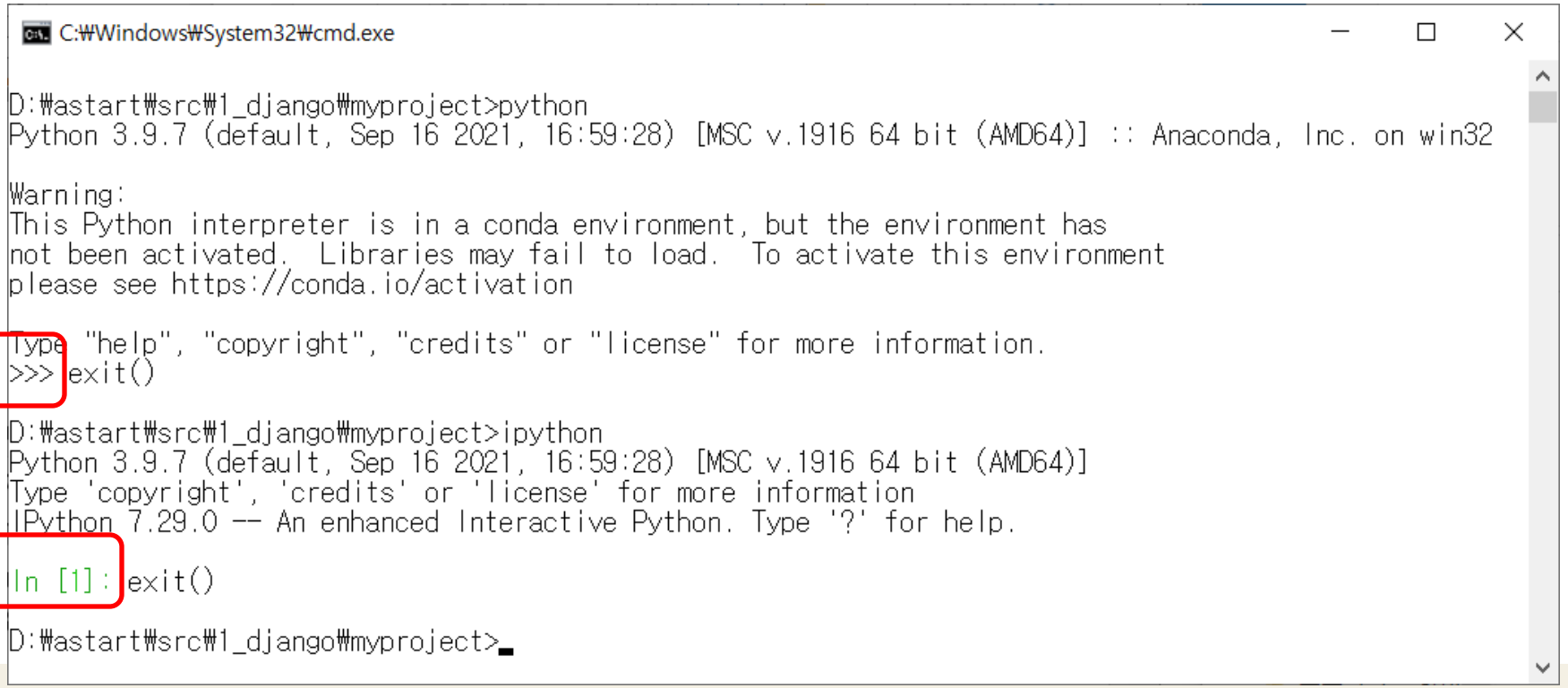
Manager & Queryset
- Managers(https://docs.djangoproject.com/ko/5.2/topics/db/managers/)
- QuerySet API reference(https://docs.djangoproject.com/en/5.2/ref/models/querysets/)
- Making queries(https://docs.djangoproject.com/en/5.2/topics/db/queries/)

1. Django Shell

Python Interactive Shell

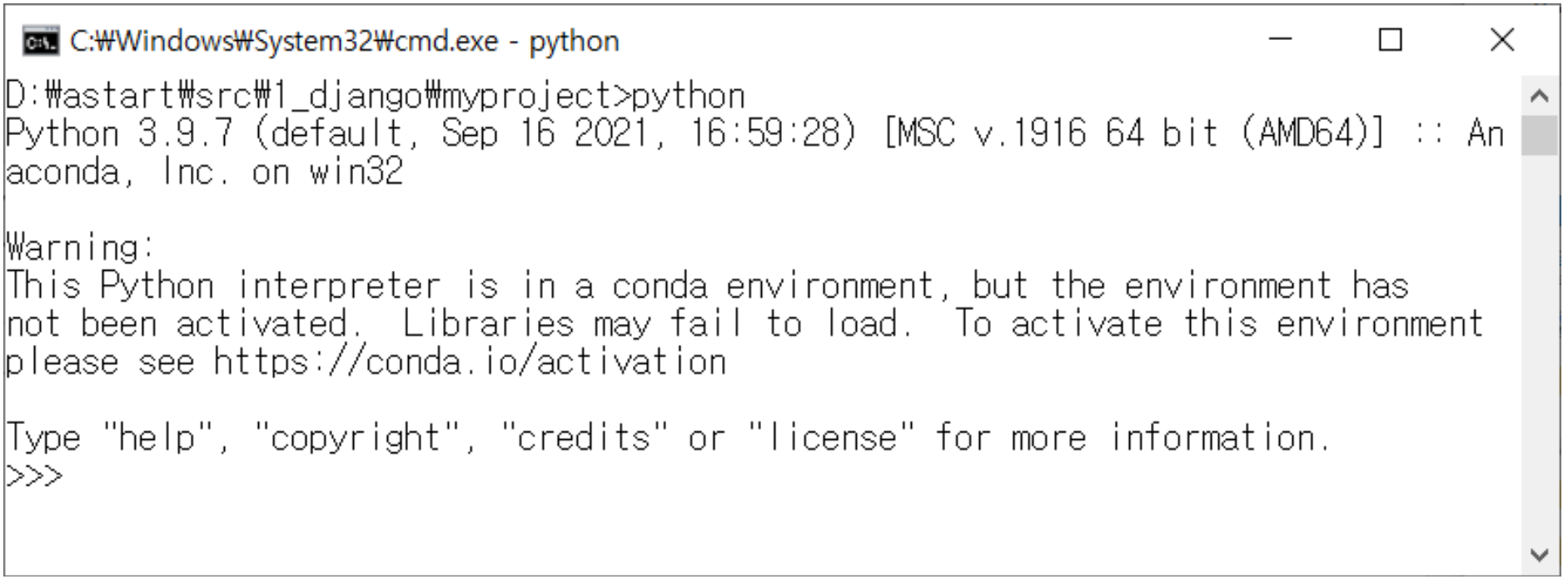
1. Django Shell

- 기본 Python shell
- IPython : https://ipython.org
- Jupyter Notebook with Python Kernel
- Bpython : https://bpython-interpreter.org



Django Shell

- 기본 파이썬 쉘



Django Shell

- Django에서 지원하는 파이썬 쉘 : 프로젝트 환경값을 가져오는 shell

```
D:\src\myproject>python manage.py shell


In [1]: from blog.models import Post

In [2]: Post.objects.all().count()

Out[2]: 6

In [3]: for post in Post.objects.all():
           ...:  print(post.id, post.title)
```

2. Manager ≒ QuerySet

Manager 클래스

- 모델의 DB query 작업을 지원하는 클래스
- 모든 모델은 반드시 하나이상의 Manager 속성을 가짐
- Manager 객체를 갖는 속성 지정

```python
from django.db import models

class Person(models.Model):
    #...
    people = models.Manager()
```

- Manager 객체를 갖는 속성을 지정하지 않으면 objects 이름으로 자동 지정
- Manager 객체 접근시 모델명.objects 으로 사용

Manager 클래스

- 모델의 전체 데이터 조회
  - 모델명.objects.all()
  - select * from app_model;
- 모델의 최근 10개 데이터 조회
  - 모델명.objects.all().order_by('-id')[:10]
  - select * from app_model order by id desc limit 10;
- 특정 모델의 새로운 데이터 추가
  - 모델명.objects.create(title="New Title")
  - insert into app_model (title) values ("New Title");
- 참조 문서
  - https://docs.djangoproject.com/ko/5.2/topics/db/managers

QuerySet

- SQL을 생성해주는 인터페이스
- 순회가능한 객체
- Manager를 통해 해당 모델의 QuerySet 객체 획득

```
>> queryset = Article.objects.all()
>> queryset
>> queryset.__class__
   django.db.models.query.QuerySet
```

- QuerySet API 참조 문서
  - https://docs.djangoproject.com/en/5.2/ref/models/querysets
- Making queries
  - https://docs.djangoproject.com/en/5.2/topics/db/queries

QuerySet

- **Chaining 지원 : 리턴값이 QuerySet인 메소드**

```
>> Article.objects.all().order_by('-id')
>> Article.objects.all().filter(..).exclude(..).filter(..)
```

- **Lazy 실행**
  - QuerySet을 만드는 시점에는 DB 접근을 하지 않음
  - 실제로 데이터가 필요한 시점을 접근 실행.
    - print(queryset)
    - list(queryset)
    - for article in queryset:
      print(article.title)

- QuerySet 실습

```
>> Article.objects.all()
>> from django.db import connection
>> connection.queries[-1]
>> Article.objects.create(title='테스트 제목',body='테스트 내용',status='d')
>> connection.queries[-1]
```

3. 조회 SQL

데이터 조회

3. 조회 SQL

- **조건을 만족하는 다수의 객체를 갖는 QuerySet 반환**
  - queryset.filter(조건) : 조건에 해당하는 데이터 조회
  - queryset.exclude(조건) : 조건에 해당하지 않는 데이터 조회

- **조건을 만족하는 하나의 객체를 반환**
  - queryset[index] : 인덱스에 해당하는 객체 또는 IndexError
  - queryset.get(조건) : 조건에 해당하는 객체 또는 DoesNotExist, MultipleObjectsReturned
  - queryset.first( ) : 첫번째 객체 또는 None
  - queryset.last() : 마지막 객체 또는 None

Filter / exclude

3. 조회 SQL

```
D:\src\myproject>python manage.py shell_plus --print-sql

In [1]: Bookmark.objects.filter(title='naver')
Out[1]: SELECT "bookmark_bookmark"."id", "bookmark_bookmark"."title",
        "bookmark_bookmark"."url" FROM "bookmark_bookmark"
       WHERE "bookmark_bookmark"."title" = 'naver'  LIMIT 21

Execution time: 0.000000s [Database: default]

<QuerySet [<Bookmark: naver>]>

In [2]: Bookmark.objects.exclude(title='naver')
Out[2]: SELECT "bookmark_bookmark"."id", "bookmark_bookmark"."title",
     "bookmark_bookmark"."url" FROM "bookmark_bookmark"
     WHERE NOT ("bookmark_bookmark"."title" = 'naver' AND
               "bookmark_bookmark"."title" IS NOT NULL)  LIMIT 21

Execution time: 0.000000s [Database: default]

<QuerySet [<Bookmark: daum>, <Bookmark: google>]>
```

Filter / exclude

3. 조회 SQL

- AND 조건
  - 방법 1

```
queryset = 모델명.objects.all()
queryset = queryset.filter(필드1=값1, 필드2=값2)
queryset = queryset.filter(필드3=값3)
queryset = queryset.filter(필드4=값4, 필드5=값5)
```

  - 방법 2

```
from django.db.models import Q
queryset.filter(Q(필드1=값1) & Q(필드2=값2))
```

- OR 조건

```
queryset.filter(Q(필드1=값1) | Q(필드2=값2))
```

Field lookup (https://docs.djangoproject.com/en/5.2/ref/models/querysets/#id2)

3. 조회 SQL

- **숫자/날짜/시간 필드**
  - 필드명__lt = 값 — 필드명 < 값
  - 필드명__lte = 값 — 필드명 <= 값
  - 필드명__gt = 값 — 필드명 > 값
  - 필드명__gte = 값 — 필드명 >= 값

- **문자열 필드**
  - 필드명__startswith = 값 — 필드명 LIKE "값%"
  - 필드명__endswith = 값 — 필드명 LIKE "%값"
  - 필드명__contains = 값 — 필드명 LIKE "%값%"
  - 필드명__istartswith =값 — 필드명 ILIKE "값%"
  - 필드명__iendswith = 값 — 필드명 ILIKE "%값"
  - 필드명__icontains = 값 — 필드명 ILIKE "%값%"

정렬조건

- 정렬 조건이 없는 경우 데이터 순서의 일관성이 없음
- 단일 정렬 조건이 다수 정렬 조건에 비해 성능에 효율적
- 시간순/역순 정렬이 필요한 경우 id 필드 활용
- 정렬 지정 방법
  - Meta 클래스에 ordering 설정

```
class Post(models.Model):

    class Meta:
        ordering = ['-updated_at']
```

  - queryset에 order_by(..) 지정

```
Article.objects.all().order_by('id')
Article.objects.all().order_by('-id')
```

범위 조건

- str, list, tuple의 슬라이싱과 비슷하지만 역순 슬라이싱은 지원하지 않음
  → DB에서 지원하지 않기 때문

- 문법
  - 객체[start:stop:step]
  - start → OFFSET
  - stop-start → LIMIT
  - step → query에 대응되지 않음

```
Article.objects.all()[10:30:2]

SELECT "article"."id", "article"."title", "article"."body",
"article"."status",
          "article"."photo" FROM "article"  LIMIT 20 OFFSET 10
```

범위 조건

- 역순 슬라이싱 처리

```
>> qs = Article.objects.all()
>>  qs[-5]
   AssertionError: Negative indexing is not supported.
>>reversed(qs.reverse()[:5])
```

데이터 조회

3. 조회 SQL ● 실습

```
## 20개의 레코드 생성
import random
for i in range(10):
  status = random.choice(['d','p','w'])
  article = Article.objects.create(title=제목#{}'.format(i),body='내용',
                                   status=status)
  print(article)
for i in range(10):
  status = random.choice(['d','p','w'])
  article = Article(title=title#{}'.format(10+i),body='내용', status=status)
  Article.save()
Article.objects.all().count()
## AND 조건
Article.objects.filter(title__icontains='t')
Article.objects.filter(title__icontains='t', title__endswith='3')
Qs1 = Article.objects.filter(title__icontains='t', title__endswith='3')
qs2 = Article.objects.filter(title__icontains='t').filter(title__endswith='
3')
print(qs1)
print(qs2)
```

데이터 조회

3. 조회 SQL

- 실습

```
## exclude( )
Article.objects.filter(title__icontains='t').exclude(title__endswith='3')

## OR 조건
from django.db.models import Q

Article.objects.filter(Q(title__icontains='1') | Q(title__endswith='3'))

Article.objects.filter(Q(title__icontains='1') & Q(title__endswith='3'))

Article.objects.filter(title__icontains='1', title__endswith='3') #위와

같음
```

데이터 조회

3. 조회 SQL

- Lazy 실행
  - DB 액세스는 실제 레코드 처리시 실행됨

```
## QuerySet 객체 생성
qs = Article.objects.filter(Q(title__icontains='1') | Q(title__endswith='3'))
qs = qs.filter(title__icontains='9')

## DB 액세스
print(qs)
qs[0]
for article in qs:
    print(article)
```

데이터 조회

3. 조회 SQL

- 실습

```
# article/url.py
from django.urls import path
from . import views

app_name = 'article'
urlpatterns = [
        path('',views.article_list),
]
```

```
# myproject/urls.py
urlpatterns = [
        path('article/',include('article.urls')),
]
```

- 실습

```
# article/views.py

from django.shortcuts import render
from .models import Article

def article_list(request):
  qs = Article.objects.all()
  q = request.GET.get('q','')
  if q:
    qs = qs.filter(title__icontains=q)

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

데이터 조회

3. 조회 SQL

- 실습

```
# article/templates/article/article_list.html

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

{% for article in article_list %}
        <li>{{article.title}} </li>
{% endfor %}
```

정렬

3. 조회 SQL

실행시 정렬 조건 지정

```
queryset = queryset.order_by("필드1")          #지정 필드 오름차순 정렬
queryset = queryset.order_by("-필드1")         #지정 필드 내림차순 정렬
queryset = queryset.order_by("필드1", "필드2")  #1차 정렬, 2차 정렬
```

기본 정렬

```
class Article(models.Model):
        ~ 생략 ~
        class Meta:
                ordering = ['id']
```

슬라이싱

3. 조회 SQL

```
## list
mynums = list(range(100))
mynums[1:10:2]                  # [1, 3, 5, 7, 9]
mynums[10:1:-2]                 # [10, 8, 6, 4, 2]

## QuerySet
queryset = Article.objects.all()
queryset[:5]                    #1~5번째 조회
queryset[5:10]                  #5~10번째 조회
queryset.order_by('-id')[:5] #내림차순 정렬후 1~5번째 조회
queryset[-5:]                   #AssertionError 예외 발생, 음수사용불가
```

DB로부터 데이터 Fetch

3. 조회 SQL

```
## 다수 레코드 fetch

for model_instance in queryset:
        print(model_instance)


## 인덱스 사용하여 1개의 레코드 fetch

model_instance = queryset[0]
model_instance = queryset[20]


## get( ) 사용하여 1개의 레코드 fetch

model_instance = queryset.get(id=1)
model_instance = queryset.get(title='제목1')
```

QuerySet 데이터 Fetch 메소드

3. 조회 SQL

- **get( )**
  - 1개의 레코드 추출
  - 0개 매칭 : ModelCls.DoesNotExist 발생
  - 1개 매칭 : 정상적 처리
  - 2개이상 매칭 : ModelCls.MultipleObjectsReturned 발생

- **first( )/last( )**
  - 결과에서 첫번째/마지막 레코드 Fetch
  - 결과값이 없는 경우 None 반환

```
model_instance = queryset.first()
model_instance = queryset.last()
```

4. 생성/수정/삭제 SQL

데이터 추가

4. 생성/수정/삭제 SQL

방법1

- 모델 객체의 save( ) 함수 사용
- 반환값 : None

```
model_instance = ModelCls(필드명1=값1,필드명2=값2)
model_instance.save()
```

방법2

- 모델 Manager의 create( ) 함수 사용
- 반환값 : 모델 객체

```
model_instance = 모델명.objects.create(필드명1=값1,필드명2=값2)
```

데이터 추가

4. 생성/수정/삭제 SQL

- **방법3**
  - ModelForm의 save( )함수 사용
  - 반환값 : 모델 객체

```
form = BookModelForm(request.POST, request.FILES)
if form.is_valid():
        book = form.save()
        book.pk
```

데이터 추가

4. 생성/수정/삭제 SQL

```
## Model의 save() 함수 사용
article = Article(title='아시안게임',body='손흥민,황의조,이승우 선수의 활약이 빛나다',status='d')
article.id
article.save()
article.id

## Manager의 create() 함수 사용
article = Article.objects.create(title='제비',body='일본으로 이동중인 태풍',status='p')
article.id
```

데이터 수정

- **방법1**
  - Model 객체의 save() 함수 사용
    - ① 수정하려는 모델 인스턴스 획득
    - ② 모델 인스턴스 속성 변경
    - ③ 모델 인스턴스의 save( )함수 실행

```
article = Article.objects.first()
article.field1 = new_value1
article.field2 = new_value2
article.save()
```

  - 모든 필드값에 대해 update가 실행됨

```
>>bookmark = Bookmark.objects.get(id='1')
>>bookmark.title = '네이버'
>>bookmark.save()
UPDATE "bookmark_bookmark" SET "title" = '네이버',
       "url" = 'http://www.naver.com' WHERE "bookmark_bookmark"."id" = 1
```

데이터 수정

4. 생성/수정/삭제 SQL

- **방법1**
  - 반환값 : None
  - 모델 인스턴스 별로 SQL문 실행
  - 다수 Row에 대해 수행시 성능저하 발생

- **방법2**
  - QuerySet의 update( ) 함수 사용
  - 하나의 SQL문으로 일괄 업데이트함
  - 반환값 : 업데이트한 Row의 개수

```
qs = Article.objects.filter(..).exclude(..)
qs.update(field1=new_value1, field2=new_value2)
```

데이터 수정

4. 생성/수정/삭제 SQL

- Model 인스턴스의 save( ) 함수 사용

```
## 하나의 row 수정
article = Article.objects.get(id=1)
article.title = '테스트 제목'
article.save()

##다수의 row 수정
queryset = Article.objects.all()
for article in queryset:
    article.status = 'p'
    article.save() #각 모델 인스턴스별로 DB에 update 요청
```

- QuerySet의 update( ) 함수 사용

```
queryset = Article.objects.all()
queryset.update(status='w') #일괄 update 처리
```

데이터 삭제

방법1

- 모델 객체의 delete( ) 함수 사용
- Model 인스턴스별로 SQL 수행
- 다수의 Row 수행시 성능저하 우려

```
article = Article.objects.all().first()
article.delete()
```

방법2

- QuerySet의 delete( ) 함수 사용
- 하나의 SQL로서 동작하므로 동작이 빠름

```
qs = Article.objects.all().filter(..).exclude(..)
qs.delete()
```

데이터 삭제

- Model 객체의 delete( ) 함수 사용

```
# 하나의 row 삭제
article = Article.objects.get(id=1)
article.delete()

# 다수의 row 삭제
qs = Article.objects.filter(title__contains='AI')
for article in qs:
    article.delete()  #각 모델인스턴스 별로 DB에 delete 요청
```

- QuerySet의 delete( ) 함수 사용

```
qs = Article.objects.filter(title__contains='AI')
qs.delete()                #일괄 delete 처리

(2, {'article.Article': 2})
```

ORM 정리

- 조회
  - all()
  - filter(조건)
  - exclude(조건)
- 추가
  1. 모델 인스턴스 생성 후, save()
  2. Manager의 create()
- 수정
  1. 모델 인스턴스 추출 후 save()
  2. QuerySet의 update() - 일괄
- 삭제
  1. 모델 인스턴스 추출 후 delete()
  2. QuerySet의 delete() – 일괄