SQL with django ORM

기본 준비 사항

```
# 폴더구조

99_sql # only SQL
    hellodb.csv
    tutorial.sqlite3
    users.csv
99_sql_orm # SQL + ORM
    ...
    users.csv # 해당 디렉토리로 다운로드
```

- django app
  - 가상환경 세팅
  - 패키지 설치
  - migrate

    ```
    $ python manage.py migrate
    $ python manage.py sqlmigrate users 0001
    ```

- `db.sqlite3` 활용
  - `sqlite3` 실행

    ```
    $ ls
    db.sqlite3 manage.py ...
    $ sqlite3 db.sqlite3
    ```

  - csv 파일 data 로드

```
sqlite > .tables
auth_group                  django_admin_log
auth_group_permissions      django_content_type
auth_permission             django_migrations
auth_user                   django_session
auth_user_groups            auth_user_user_permissions
users_user
sqlite > .mode csv
sqlite > .import users.csv users_user
sqlite > SELECT COUNT(*) FROM users_user;
100
```

- 확인
  - sqlite3에서 스키마 확인

```
sqlite > .schema users_user
CREATE TABLE IF NOT EXISTS "users_user" ("id" integer NOT NULL PRIMARY
KEY AUTOINCREMENT, "first_name" varchar(10) NOT NULL, "last_name"
varchar(10) NOT NULL, "age" integer NOT NULL, "country" varchar(10) NOT
NULL, "phone" varchar(15) NOT NULL, "balance" integer NOT NULL);
```

문제

> 아래의 문제들을 보면서 서로 대응되는 ORM문과 SQL문을 작성하시오.
>
> **vscode 터미널을 좌/우로 나누어 진행하시오. (sqlite / shell_plus)**

`.headers on` 만 켜고 작성해주세요.

1. 기본 CRUD 로직

1. 모든 user 레코드 조회

```
# orm
User.objects.all()
```

```
-- sql
SELECT * FROM users_user;
```

2. user 레코드 생성

```
# orm
User.objects.create(
    first_name='길동',
    last_name='홍',
    age=100,
    country='제주도',
    phone='010-1234-5678',
    balance=10000
)
```

```
-- sql
INSERT INTO "users_user" ("first_name", "last_name", "age", "country",
"phone", "balance")
VALUES ('길동', '홍', 100, '제주도', '010-1234-1234', 100)
```

- 하나의 레코드를 빼고 작성 후 `NOT NULL` constraint 오류를 orm과 sql에서 모두 확인 해보세요.

3. 해당 user 레코드 조회
   - `102` 번 id의 전체 레코드 조회

```
# orm
User.objects.get(pk=102)
```

```
-- sql
SELECT * FROM users_user WHERE id = 102;
```

4. 해당 user 레코드 수정
   - ORM: `102` 번 글의 `last_name` 을 '김' 으로 수정
   - SQL: `102` 번 글의 `first_name` 을 '철수' 로 수정

```
# orm
user = User.objects.get(pk=102)
user.last_name = '김'
user.save()
user.last_name
```

```
-- sql
UPDATE users_user SET first_name='철수' WHERE id=102;
SELECT * FROM users_user WHERE id=102;
```

5. 해당 user 레코드 삭제
   - ORM: `102` 번 글 삭제
   - `SQL`: `101` 번 글 삭제

```
# orm
User.objects.get(pk=102).delete()
```

```sql
-- sql
DELETE FROM users_user WHERE id=101;
SELECT * FROM users_user WHERE id=101;
```

---

2. 조건에 따른 쿼리문

1. 전체 인원 수
    - `User` 의 전체 인원수

```python
# orm
User.objects.count()
```

```sql
-- sql
SELECT COUNT(*) FROM users_user;
```

2. 나이가 30인 사람의 이름
    - `ORM` : `.values` 활용
        - 예시: `User.objects.filter(조건).values(컬럼이름)`

```python
# orm
User.objects.filter(age=30).values('first_name')
```

```sql
-- sql
SELECT first_name FROM users_user WHERE age=30;
```

3. 나이가 30살 이상인 사람의 인원 수
    - ORM: `__gte` , `__lte` , `__gt`, `__lt` -> 대소관계 활용

```python
# orm
User.objects.filter(age__gte=30).count()
```

```sql
-- sql
SELECT COUNT(*) FROM users_user WHERE age>=30;
```

4. 나이가 20살 이하인 사람의 인원 수

```python
# orm
User.objects.filter(age__lte=20).count()
```

```sql
-- sql
SELECT COUNT(*) FROM users_user WHERE age<=20;
```

5. 나이가 30이면서 성이 김씨인 사람의 인원 수

```
# orm
User.objects.filter(age=30, last_name='김').count()

# filter 두개 걸 수도 있음
User.objects.filter(age=30).filter(last_name='김').count()
```

```
-- sql
SELECT COUNT(*) FROM users_user WHERE age=30 AND last_name='김';
```

6. 나이가 30이거나 성이 김씨인 사람? **from django.db.models import Q**

```
# orm
from django.db.models import Q  # 똑같이 shell_plus 켜진 cmd 창에 입력하면 됨
User.objects.filter(Q(age=30) | Q(last_name='김'))
```

```
-- sql
SELECT * FROM users_user WHERE age=30 OR last_name='김';
```

7. 지역번호가 02인 사람의 인원 수
   - `ORM`: `__startswith`

```
# orm
User.objects.filter(phone__startswith='02-').count()
```

```
-- sql
SELECT COUNT(*) FROM users_user WHERE phone LIKE '02-%';
```

8. 거주 지역이 강원도이면서 성이 황씨인 사람의 이름

```
# orm
User.objects.filter(country='강원도', last_name='황').values('first_name')
```

```
-- sql
SELECT first_name FROM users_user WHERE country='강원도' AND last_name='황';
```

---

3. 정렬 및 LIMIT, OFFSET

1. 나이가 많은 사람순으로 10명

```
# orm
User.objects.order_by('-age')[:10]
```

```
-- sql
SELECT * FROM users_user ORDER BY age DESC LIMIT 10;
```

2. 잔액이 적은 사람순으로 10명

```
# orm
User.objects.order_by('balance')[:10]
```

```
-- sql
SELECT * FROM users_user ORDER BY balance ASC LIMIT 10;
```

3. 잔고는 오름차순, 나이는 내림차순으로 10명?

```
# orm
User.objects.order_by('balance','-age')[:10]
```

```
-- sql
SELECT * FROM users_user ORDER BY balance ASC, age DESC LIMIT 10;
```

4. 성, 이름 내림차순 순으로 5번째 있는 사람

```
# orm
User.objects.order_by('-last_name', '-first_name')[4]
# 0부터 시작
```

```
-- sql
SELECT * FROM users_user ORDER BY last_name DESC, first_name DESC LIMIT 1
OFFSET 4;
```

---

4. 표현식

> ORM: `aggregate` 사용
>
> https://docs.djangoproject.com/en/3.2/topics/db/aggregation/#aggregation
>
> - '종합', '합계' 등의 사전적 의미
> - 특정 필드 전체의 합, 평균 등을 계산할 때 사용

1. 전체 평균 나이

```
# orm
from django.db.models import Avg
User.objects.aggregate(Avg('age'))
```

```
-- sql
SELECT AVG(age) FROM users_user;
```

2. 김씨의 평균 나이

```
# orm
User.objects.filter(last_name='김').aggregate(Avg('age'))
```

```
-- sql
SELECT AVG(age) FROM users_user WHERE last_name='김';
```

3. 강원도에 사는 사람의 평균 계좌 잔고

```
# orm
User.objects.filter(country='강원도').aggregate(Avg('balance'))
```

```
-- sql
SELECT AVG(balance) FROM users_user WHERE country='강원도';
```

4. 계좌 잔액 중 가장 높은 값

```
# orm
User.objects.aggregate(Max('balance'))
```

```
-- sql
SELECT MAX(balance) FROM users_user;
```

5. 계좌 잔액 총액

```
# orm
User.objects.aggregate(Sum('balance'))
```

```
-- sql
SELECT SUM(balance) FROM users_user;
```

6. Annotate 칼럼에 주석 달아서 결과 보기

```
# orm
from django.db.models import Count

User.objects.values('country').annotate(Count('country'))
# <QuerySet [{'country': '강원도', 'country__count': 14}, ...]>

User.objects.values('country').annotate(num_countries=Count('country'))
# <QuerySet [{'country': '강원도', 'num_countries': 14}, ...]>

User.objects.values('country').annotate(Count('country'),
avg_balance=Avg('balance'))
# <QuerySet [{'country': '강원도', 'country__count': 14, 'avg_balance':
157895.0}, ... ]>
```

```
-- sql
sqlite> SELECT country, COUNT(country) FROM users_user GROUP BY country;
```