SQLite 정리

우리는 Django ORM 을 사용해 데이터베이스를 만들어서 활용했으나,

사실 ORM 이라는 것은 프레임워크에서 여러 DB 를 똑같은 문법으로

사용하기 위해 지원하는 언어이다.

ORM 은 결국 SQL 이라는, 각각의 DB 에 사용되는 언어로 번역된다.

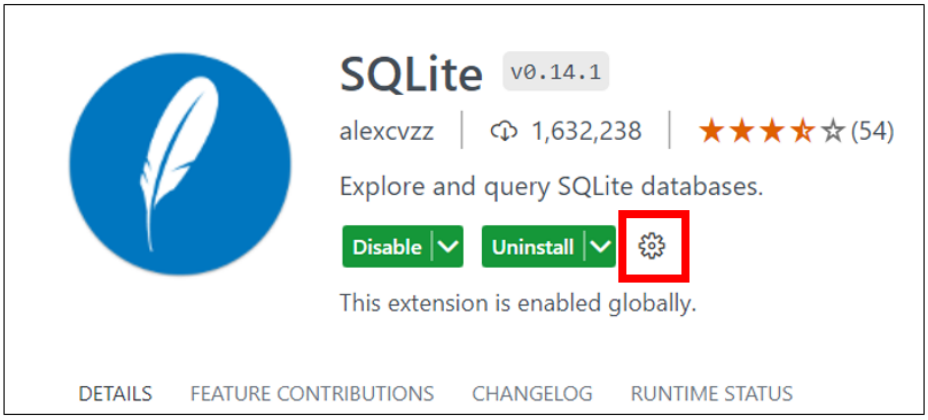
우리가 사용하는 DB 인 SQLite 의 SQL 에 대해 배워보는 시간을 가진다.

0. Get Started

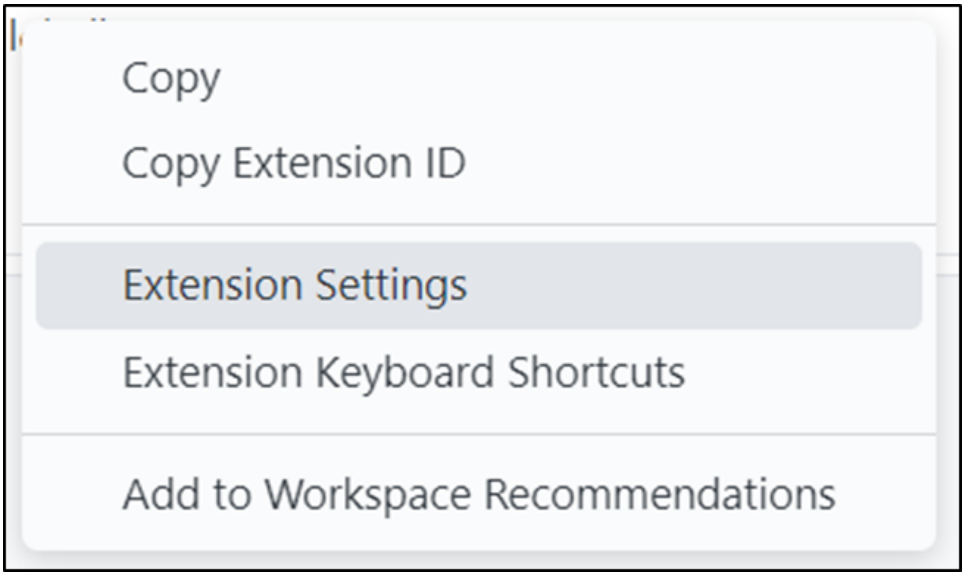
원래는 chocolatey 를 사용해 SQLite 를 설치하려고 했으나 (생략하자 우리는 어제 이미 설치 했다.)
참고로 chocolatey 는 윈도우 환경에서 사용하는 커맨드라인 패키지 매니저 프로그램이다.
프로그램 설치하면 환경변수를 자동으로 셋팅해 주는 소프트웨어 인데 궁금하신 분은
google 검색 해봐라. (우리는 어제 환경변수설정 까지 공식문서를 보고 직접 다 했다.)

여기서 부터 보자.

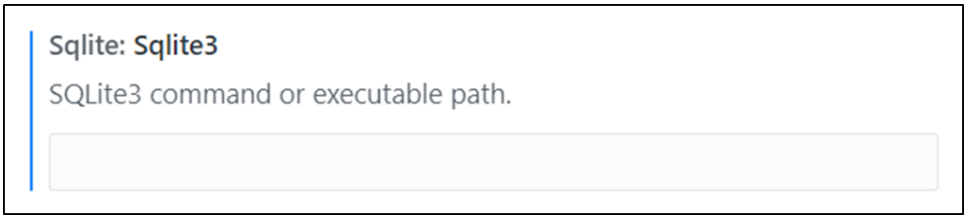
VSCode 에서 sqlite 익스텐션을 설치한다.



설치한 후, 톱니바퀴를 클릭하자.



익스텐션 세팅 클릭



sqlite3 라고 되어있는 칸을 “빈 칸” 으로 만든다. 그 이유는 sqlite3 명령줄을 기본으로 설정할 시

익스텐션이 제대로 작동 안하기 때문이다. 무슨 문제인지는 불명. (파악중)

VSCode 를 끄고, 바탕화면에 실습을 위한 디렉터리를 아무거나 하나 만든 후,

VSCode 를 해당 경로 기준으로 작동 시키고, 다음 두 개의 파일을 만들자.

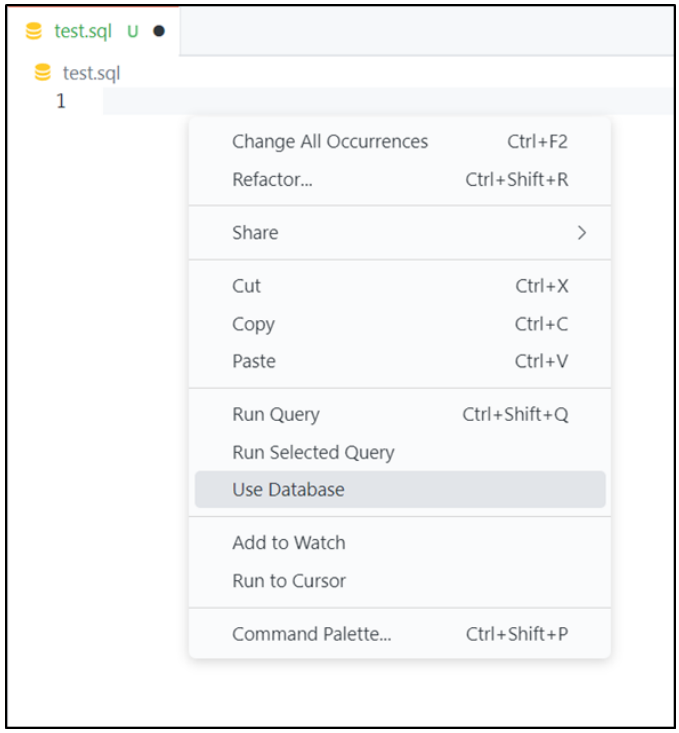
이름은 뭐가 되었든 상관은 없는데, 확장자는 `.sql` 과, `.sqlite3` 로 두 개를 정확히 써줘야 한다.

`.sql` : SQL 을 작성하고 실행시키는 파일 (sql문을 적을 연습장)

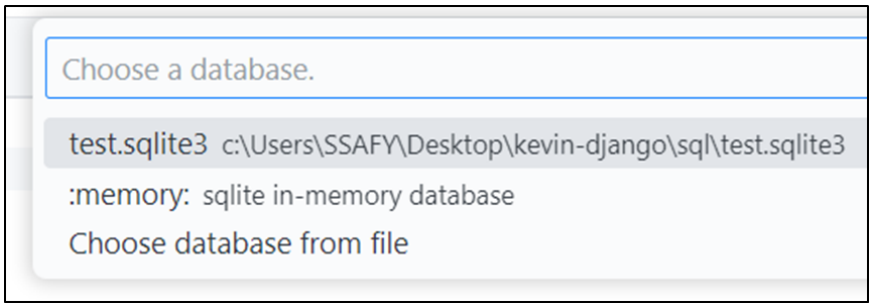
`.sqlite3` : SQLite DB 를 말한다.

다른 RDBMS 와 다른 SQLite 의 가장 큰 특징은, 하나의 DB 가 하나의 “파일” 이라는 것이다.

`test.sql` 을 열고, 파일의 빈 공간에 오른쪽 마우스를 클릭한다.



Use Database 클릭



`test.sqlite3` 클릭

이제, 다음과 같은 테이블을 만들고 싶다고 가정하겠다.

name	age	address
jony	30	서울
sylvie	24	대구
nana	4	광주

엑셀에서 흔히 보던 표 아닌가? 그렇다. 테이블은 별 거 없다. 그냥 표다.

이를 위해 `test.sql` 에 다음과 같이 작성한다. `test.sqlite3` 에 작성 하는 것이 아니다!

```
CREATE TABLE classmates (
    name TEXT NOT NULL,
    age INTEGER NOT NULL,
    address TEXT NOT NULL
);

INSERT INTO classmates (name, age, address)
VALUES
    ('jony', 30, '서울'),
    ('sylvie', 24, '대구'),
    ('nana', 4, '광주');
```

문법을 알아보자.

먼저, `CREATE TABLE` 은 테이블을 만들 때 쓴다.
하나의 DB 파일은 여러 개의 “TABLE” 이라는 단위로 구성된다.

총 세 개의 컬럼(column) 으로 구성되어있으며, 각각 `name` , `age` , `address` 이다.
필드 다음 나오는 `TEXT` , `INTEGER` 는 무엇을 의미하나? 데이터 타입을 의미한다.

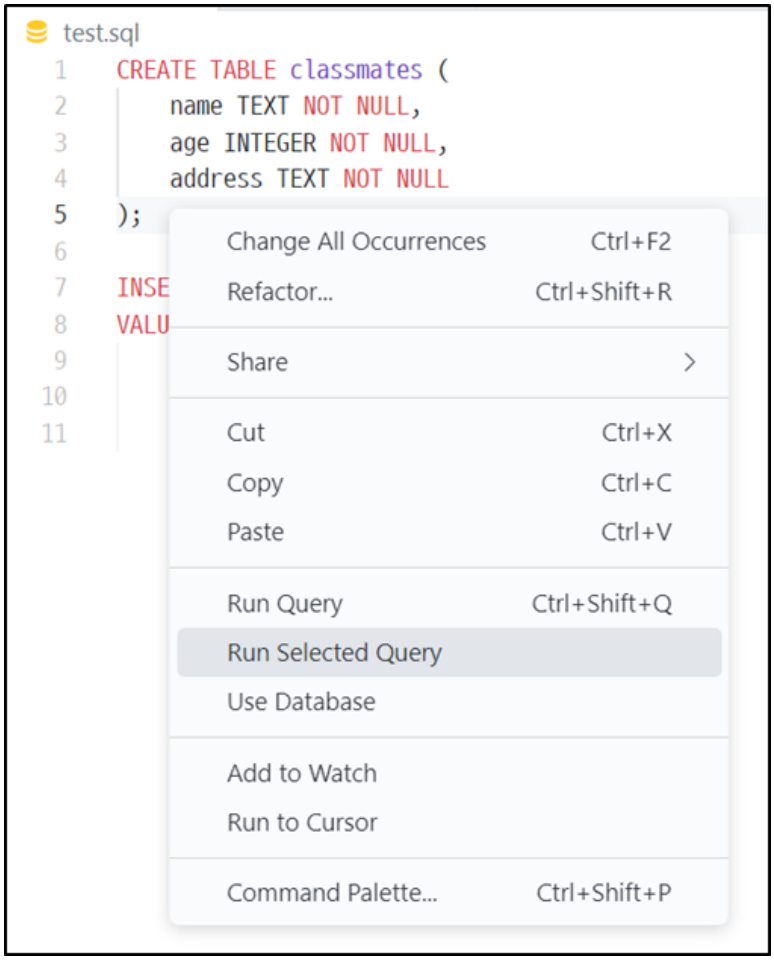
`NOT NULL` 은 공백 허용 안함이다. `NOT NULL` 이 있다면 반드시 데이터가 들어가야 한다.
각 컬럼의 행의 끝은 콤마 `,` 로 끝나고, 맨 끝 컬럼엔 콤마를 붙여주지 않는다.

모든 컬럼을 소괄호 () 로 감싼 다음, 맨 마지막엔 반드시 세미콜론 `;` 을 붙여준다.
이렇게 만들어진 테이블 안에는 실제 데이터가 존재하지 않는다.

`INSERT INTO` 는 로우 `row` 를 생성할 때, 즉 실제 데이터를 생성할 때 사용한다.
`'jony'` , `'sylvie'` , `'nana'` 는 `name` 컬럼에 해당하며, `30` , `24` , `4` 는 `age` 컬럼에 해당한다. `'서울'` , `'대구'` , `'광주'` 는 `address` 컬럼에 해당한다.

이렇게 작성된 문법이 바로 SQL, 쿼리문이라고 한다. 실행단위는 세미콜론 `;` 이며, VSCode 에 한 번에 실행되지 않는다.

실행법은, 각 세미콜론의 마치는 지점을 클릭해서 커서를 활성화시키고,



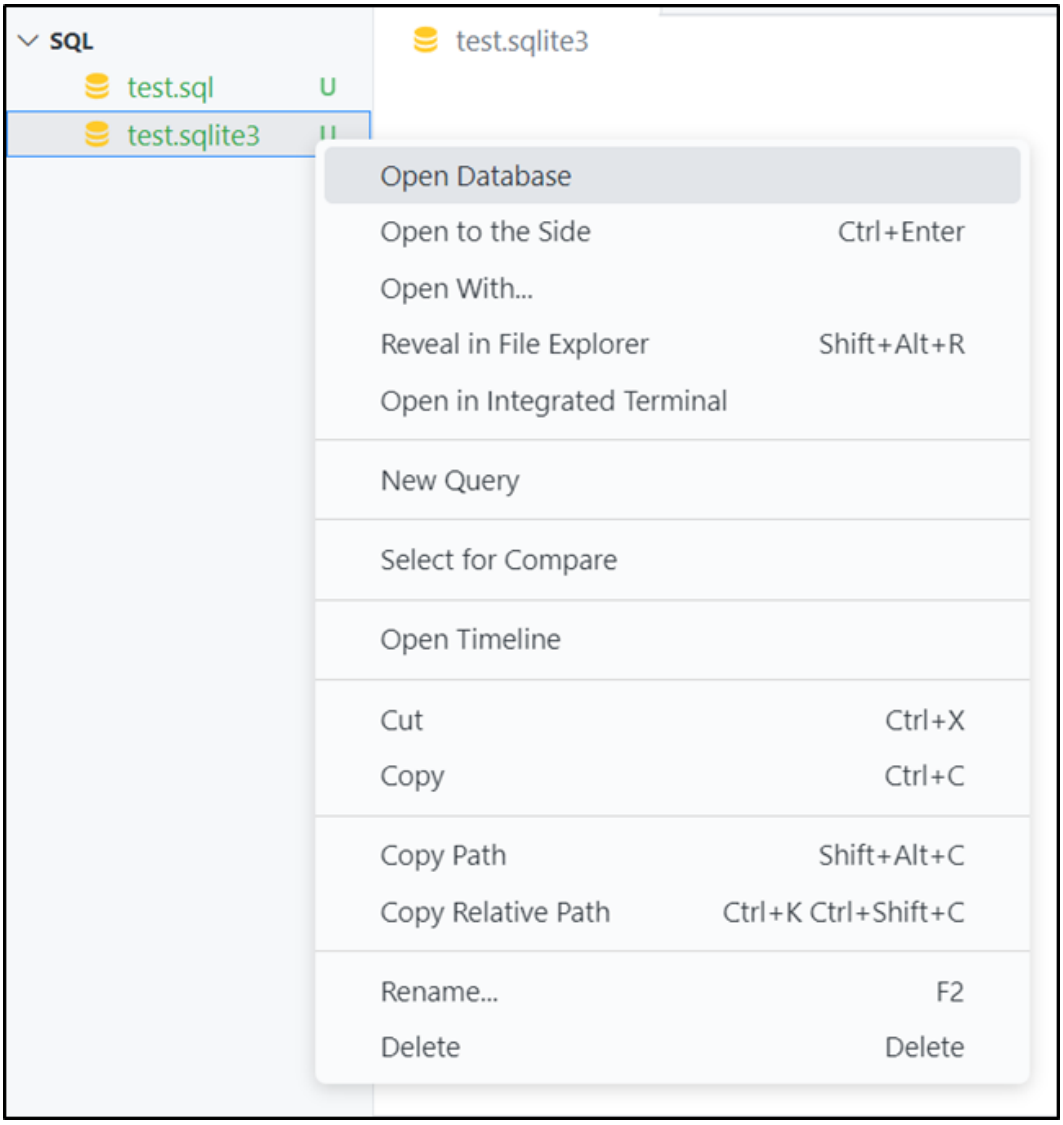
오른쪽 버튼을 클릭해 `Run Selected Query` 를 실행시킨다.

즉, 두 번 해야한다. `CREATE TABLE` 의 끝지점, `INSERT INTO` 의 끝지점.

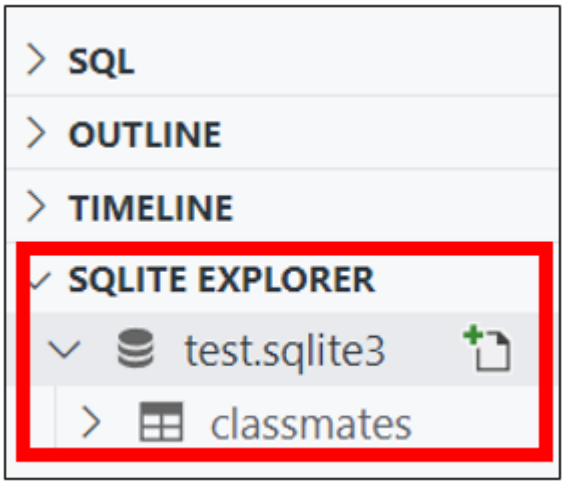
그러면 오른쪽에 다음과 같이 빈 SQL화면이 뜬다.

이건 꺼버리고, 왼쪽 네비게이션 바에서 `test.sqlite3` , 즉 DB 파일을 오른쪽 클릭한다.

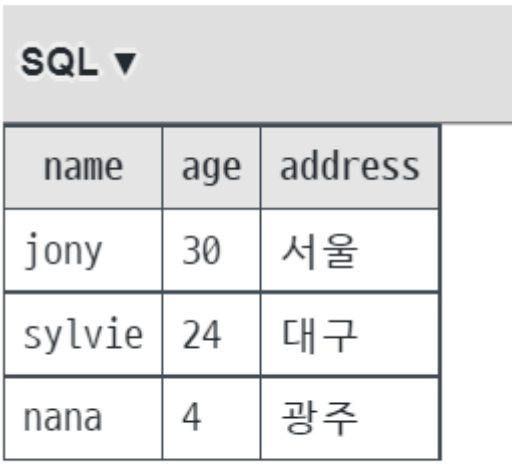
헷갈리지 말자. `.sql` 은 쿼리문을 작성하는 파일이고, `.sqlite3` 가 실제 DB다.



여기서 `Open Database` 를 클릭하면,



네비게이션바 아래, 즉 VSCode 화면 왼쪽 아래에 `SQLITE EXPLORER` 가 활성화되고,



`classmates` 테이블의 재생 버튼을 클릭하면 다음과 같은 테이블을 확인할 수 있다.

다음 명령을 사용해 테이블을 삭제하도록 하자.

```
DROP TABLE classmates;
```

쿼리문 실행 후, SQLITE EXPLORER 의 새로고침을 클릭해보자.

테이블 `classmates` 가 삭제된 것을 확인할 수 있다.

1. `CREATE TABLE`

테이블 생성 구문에 대해 배워보자.

```
CREATE TABLE classmates (
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    email TEXT NOT NULL UNIQUE
);
```

1-1. 데이터 타입

현재 데이터타입은 `TEXT` 와 `INTEGER` 만 확인되며, 자주 쓰는 데이터 타입은 다음과 같다.

`INTEGER` : 정수형. `123`

`REAL` : 실수형. `25.84`

`TEXT` : 문자열 `'하이'`

꼭 있어야 할 것 중, SQLite 에는 `BOOLEAN` 타입이 없다. SQLite 는 True, False 를 따로 저장하진 않고, `INTEGER` 타입에 `0` , `1` 을 저장해서 참 거짓을 나타낸다.

또한, `DATE` , `TIME` 또한 존재하지 않는데, 이는 차후에 함수로 저장한다.

`VARCHAR` 처럼 문자열의 길이를 정하는 타입은 존재하지 않는데

SQLite 는 문자열이라면 길든 짧든 모두 그냥 `TEXT` 로 처리하기 때문이다.

문자열의 경우, 작은 따옴표 하나 `'` 로 감싼다.

- Django Model 을 작성하다보면 분명 길이도 정할 수 있고, 날짜, 시간, 참 거짓도 가능한데, Django에서 ORM 을 통해 SQLite 에 보내는 SQL 은 Django ORM이 알아서 적절히 번역해줘서

  날짜 시간 참 거짓 등이 가능 했던 것이다. 원래의 SQLite 에선 해당 타입들은 존재하지 않는다.

1-2. `PRIMARY KEY`, `AUTOINCREMENT`, `UNIQUE`

데이터 타입 이외에 `PRIMARY KEY` , `AUTOINCREMENT` , `UNIQUE` 가 추가되었다.

`PRIMARY KEY` : 주키, PK

`AUTOINCREMENT` : 데이터 추가 시 1 씩 자동 증가

`UNIQUE` : 중복 허용 안함. 어떤 사람이 `sfminhol@gmail.com` 을 가지고 있다면, 다른 사람은 동일한 이름의 이메일을 사용할 수 없다.

`age` 컬럼의 경우, `NOT NULL` 이 붙어있지 않기 때문에, `NULL` 값으로 세팅 가능하다.

`Run Selected Query` 를 사용해 실행하고, 다음 샘플 데이터를 입력해 결과를 확인해보자.

```
INSERT INTO classmates (name, age, email)
VALUES
    ('jony', 30, 'jony@email.com'),
    ('sylvie', 24, 'tarrie@hotmail.com'),
    ('nana', 4, 'nanana@naver.com');
```

id	name	age	email
1	jony	30	jony@email.com
2	sylvie	24	tarrie@hotmail.com
3	nana	4	nanana@naver.com

분명 `id` 컬럼은 기입한 적 없는데, 1 부터 자동으로 증가되는것을 확인할 수 있다.

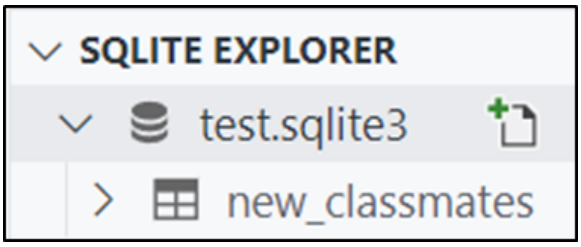
2. `ALTER TABLE`

테이블 자체를 수정할 때 사용한다.

2-1. `RENAME TO`

`contacts` 테이블의 이름을 `new_contacts` 로 바꿔보자.

```
ALTER TABLE classmates RENAME TO new_classmates;
```



2-2. `RENAME COLUMN`

`name` 컬럼을 `username` 으로 바꿔보자.

```
ALTER TABLE new_classmates RENAME COLUMN name TO username;
```

id	username	age	email
1	jony	30	jony@email.com
2	sylvie	24	tarrie@hotmail.com
3	nana	4	nanana@naver.com

2-3. `ADD COLUMN`

사용자 주소를 받기 위해 `address` 라는 컬럼을 추가해보자. 널값은 허용 안할 것이다.

```
ALTER TABLE new_classmates ADD COLUMN address TEXT NOT NULL DEFAULT 'no address';
```

id	username	age	email	address
1	jony	30	jony@email.com	no address
2	sylvie	24	tarrie@hotmail.com	no address
3	nana	4	nanana@naver.com	no address

널값을 허용안한다면, 기존에 존재하는 데이터들은 어떤 `address` 를 가지는지 알 수 없으므로 반드시 `DEFAULT` 세팅할 값을 정해주어야한다. 지금 예시에선 `'no address'` 로 세팅했다.

2-4. `DROP COLUMN`

컬럼을 지우는 건 `sqlite` 를 터미널에서 직접 켜서 타이핑 해야만 작동되는 것을 확인 할 수 있을 것이다.

```
$ sqlite3 test.sqlite3
```

그리고 `sqlite` 셸에서 다음과 같이 입력

```
ALTER TABLE new_classmates DROP COLUMN address;
```

이후 테이블을 확인해보자.

id	username	age	email
1	jony	30	jony@email.com
2	sylvie	24	tarrie@hotmail.com
3	nana	4	nanana@naver.com

`address` 컬럼이 삭제되었다.

실습 종료를 위해 테이블을 삭제하자.

```
DROP TABLE new_classmates;
```

3. `.csv`

우선 다음 파일을 다운로드받아 프로젝트 디렉터리 안에 넣는다.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c1db62af-cb7b-4a88-8347-0325b4eab895/users.csv

우리가 설치한 SQLite Windows 버전은 아직 한 번도 사용한 적이 없다.

터미널에 다음과 같이 입력해 `test.sqlite3` 파일을 SQLite 를 사용해 실행시키자.

```
$ sqlite3 test.sqlite3

SQLite version 3.41.1 2023-03-10 12:13:52
Enter ".help" for usage hints.
sqlite>
```

다음과 같이 `sqlite>` 가 보이면 성공이다. 이것을 SQLite Shell 이라고 하겠다.

열었던 SQLite Shell 을 닫을 땐 `.exit` 또는 `.quit` 을 입력한다.

(열기 sqlite3 test.sqlite3)

SQLite Shell 을 사용해 우리가 다운로드받은 `users.csv` 파일을 DB 에 입력할 것이다.

잠깐 열어보면 다음과 같은 형태인데,

```
정호,유,40,전라북도,016-7280-2855,370
경희,이,36,경상남도,011-9854-5133,5900
정자,구,37,전라남도,011-4177-8170,3100
미경,장,40,충청남도,011-9079-4419,250000
영환,차,30,충청북도,011-2921-4284,220
서준,이,26,충청북도,02-8601-7361,530
주원,민,18,경기도,011-2525-1976,390
예진,김,33,충청북도,010-5123-9107,3700
...
```

이 데이터를 넣기 위해 우선 `.csv` 파일의 현재 형태에 맞는 테이블이 존재해야한다.

다음과 같은 쿼리문을 `test.sql` 에 작성해 실행한다.

```
CREATE TABLE users1 (
  first_name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  age INTEGER NOT NULL,
  country TEXT NOT NULL,
  phone TEXT NOT NULL,
  balance INTEGER NOT NULL
);
```

그리고 터미널을 열고, 다음과 같이 입력한다.

```
$ sqlite3 test.sqlite3

sqlite> .mode csv
sqlite> .import users.csv users1
```

그리고 DB 를 확인해보면 다음과 같다.

first_name	last_name	age	country	phone	balance
정호	유	40	전라북도	016-7280-2855	370
경희	이	36	경상남도	011-9854-5133	5900
정자	구	37	전라남도	011-4177-8170	3100
미경	장	40	충청남도	011-9079-4419	250000
영환	차	30	충청북도	011-2921-4284	220
서준	이	26	충청북도	02-8601-7361	530
주원	민	18	경기도	011-2525-1976	390

4. SELECT

테이블을 확인할 땐 `SELECT` 구문을 사용한다.

다음 쿼리문을 실행해보자.

```
SELECT * FROM users1 limit 20;
```

`*` 은 모든 컬럼을 의미하며, `limit` 는 보여줄 행의 갯수이다.

무려 1000 명의 데이터가 저장되어 있기 때문에, `limit` 를 붙이지 않으면 효율적이지못하다.

`*` 이 들어가는 곳이 컬럼명이라면, 다음과 같이 작성할수도 있다.

```
SELECT first_name, age FROM users1 limit 20;
```

`users` 테이블에서 `first_name` 과 `age` 컬럼만 보여달라는 뜻이다.

first_name	age
정호	40
경희	36
정자	37
미경	40
영환	30
서준	26
주원	18
예진	33

4-1. `ORDER BY`

정렬은 `ORDER BY` 를 사용한다.

```
SELECT * FROM users1 ORDER BY balance limit 20;
```

현재 사용자의 `balance` 기준 정렬이다.

first_name	last_name	age	country	phone	balance
우진	성	32	전라북도	010-7636-4368	150
광수	김	32	충청남도	02-7695-5421	150
민재	오	20	전라남도	011-2524-3749	150
승민	김	28	경기도	010-2919-6864	150
진우	김	33	경기도	02-1114-4892	150
순옥	성	23	제주특별자치도	011-4667-2759	160
수민	김	36	경상북도	016-3424-9936	160
명자	김	32	전라남도	016-2871-7505	160

만약, 내림차순으로 정렬하고 싶다면 `DESC` 옵션을 붙여준다. (오름차순은 `ASC` 이며 기본 값이다)

```
SELECT * FROM users1 ORDER BY balance DESC limit 20;
```

first_name	last_name	age	country	phone	balance
순옥	김	24	제주특별자치도	016-4846-2896	1000000
상철	우	22	충청북도	02-1089-6693	1000000
진호	민	36	경상남도	02-5497-1785	1000000
재호	이	20	전라북도	010-3059-8786	1000000
민준	강	21	제주특별자치도	010-3698-2359	1000000
은정	황	17	전라남도	02-5773-7460	1000000
영수	김	34	충청북도	010-6210-5336	1000000
정남	허	25	경상북도	011-2737-2293	1000000

4-2. `WHERE`

`WHERE` 은 조건을 의미한다. `=` , `<` , `<=` , `>=` , `>` , `!=` 을 사용 가능하다.

나이가 30 살 이상인 사람만 출력해보자.

```
SELECT first_name, age, balance FROM users1 WHERE age >= 30;
```

first_name	age	balance
정호	40	370
경희	36	5900
정자	37	3100
미경	40	250000
영환	30	220
예진	33	3700
하은	32	35000
영일	35	720

나이 30세 이상이면서, 잔고가 50만원 초과인 사람도 찾아보고싶다. 이 경우, `AND` 를 사용할 수 있다.

- 만약 `OR` 이면? 나이가 30세 이상이거나, 잔고가 50만원 초과인 사람을 뜻한다.

```
SELECT first_name, age, balance FROM users1 WHERE age >= 30 AND balance > 500000;
```

first_name	age	balance
성현	40	580000
은주	38	950000
선영	37	570000
중수	38	780000
미경	39	890000

```
SELECT first_name, last_name, age FROM users1 WHERE first_name LIKE '%호';
```

이름이 '호' 로 끝나는 사람들의 이름과 성 조회하기

패턴 의미

패턴	의미
2%	2로 시작하는 패턴
%2	2로 끝나는 패턴
%2%	2를 포함하는 패턴
_2%	첫번째 자리에 아무 값이 하나 있고 두 번째가 2로 시작하는 패턴 (최소 2자리)
1___	1로 시작하는 4자리 패턴 (반드시 4자리)
2_%_% or 2__%	2로 시작하고 최소 3자리인 패턴 (3자리 이상일 경우)

직접 이것저것 테스트도 해보고 검색도 해보자 !

```
SELECT first_name, phone FROM users1 WHERE phone LIKE '02-%';
```

```
SELECT first_name, age FROM users1 WHERE age LIKE '2_';
```

```
SELECT first_name, country FROM users1 WHERE country IN ('경기도','강원도');
```

```
SELECT first_name, country FROM users1 WHERE country NOT IN ('경기도','강원도');
```

4-3. BETWEEN

값이 범위안에 있는지 테스트를 할때 사용

값이 지정된 범위에 있으면 true 반환 하며 결과를 부정하고 싶으면 NOT BETWEEN 연산자를 사용할 수 있다.

```
SELECT first_name, age FROM users1 WHERE age BETWEEN 20 AND 30;
```

	구문	예시
C	INSERT	**INSERT INTO** 테이블이름 (컬럼1, 컬럼2, ...) **VALUES** (값1, 값2);
R	SELECT	**SELECT** * **FROM** 테이블이름 **WHERE** 조건;
U	UPDATE	**UPDATE** 테이블이름 **SET** 컬럼1=값1, 컬럼2=값2 **WHERE** 조건;
D	DELETE	**DELETE FROM** 테이블이름 **WHERE** 조건;

<끝>

내려고 했으나 추가내용 고고

4-4. `Aggregate Function`

Aggregate Function (집계함수) ⇒ select 구문에서만 사용된다.

예> COUNT // AVG // MAX // MIN

```
SELECT COUNT(컬럼) FROM 테이블이름;
```

레코드의 개수를 조회하기

users1 테이블의 레코드 총 개수를 조회한다면?

```
SELECT COUNT(*) FROM users1;
```

```
SELECT AVG(컬럼) FROM 테이블이름;
SELECT SUM(컬럼) FROM 테이블이름;
SELECT MIN(컬럼) FROM 테이블이름;
SELECT MAX(컬럼) FROM 테이블이름;
```

위 함수들은 기본적으로 해당 컬럼이 INTEGER 일때만 사용이 가능하다

만약에 30살 이상인 사람들의 평균 나이는?

```
SELECT AVG(age) FROM users1 WHERE age>=30;
```

계좌 잔액 (blance)가 가장 높은 사람과 그 액수를 조회 하려면?

```
SELECT first_name, MAX(balance) FROM users1;
```

나이가 30 이상인 사람의 계좌 평균 잔액을 조회 하려면?

```
SELECT AVG(balance) FROM users1 WHERE age>=30;
```

4-5. `Group by`

만약에 users1 에서 각 성(last name)씨가 몇 명씩 있는지 조회 한다면??

```
SELECT last_name, COUNT(*) FROM users1 GROUP BY last_name;
```

예시를 살펴보자

SELECT 컬럼1, aggregate_function FROM 테이블 GROUP BY 컬럼1;

즉 선택된 행 그룹을 하나 이상의 열값으로 요약하여 행으로 만든다.

단 주의 점은, 문장에 WHERE 절이 있으면 반드시 WHERE절 뒤에 작성해야 한다.

아래 코드와 위에있는 코드랑 다른점을 보자. AS 를 활용했다.

```
SELECT last_name, COUNT(*) AS name_count FROM users1 GROUP BY last_name;
```

```
SELECT last_name, COUNT(*) AS name_count FROM users GROUP BY last_name;
```

AS 를 활용해서 COUNT에 해상하는 컬럼 명을 바꿔서 조회 할 수 있다.

<끝>