Day1 SQL 1,2

자료	DB
구분	DB

데이터 베이스에 대해서 학습할 것이다.

데이터 베이스란→ 데이터를 조작하고 관리하는 것을 말한다.

이러한 데이터 베이스 안의 데이터를 잘 사용, 즉 CRUD 하기 위해서

프로그램을 사용하는데 그러한 프로그램들은 DBMS라고 한다.

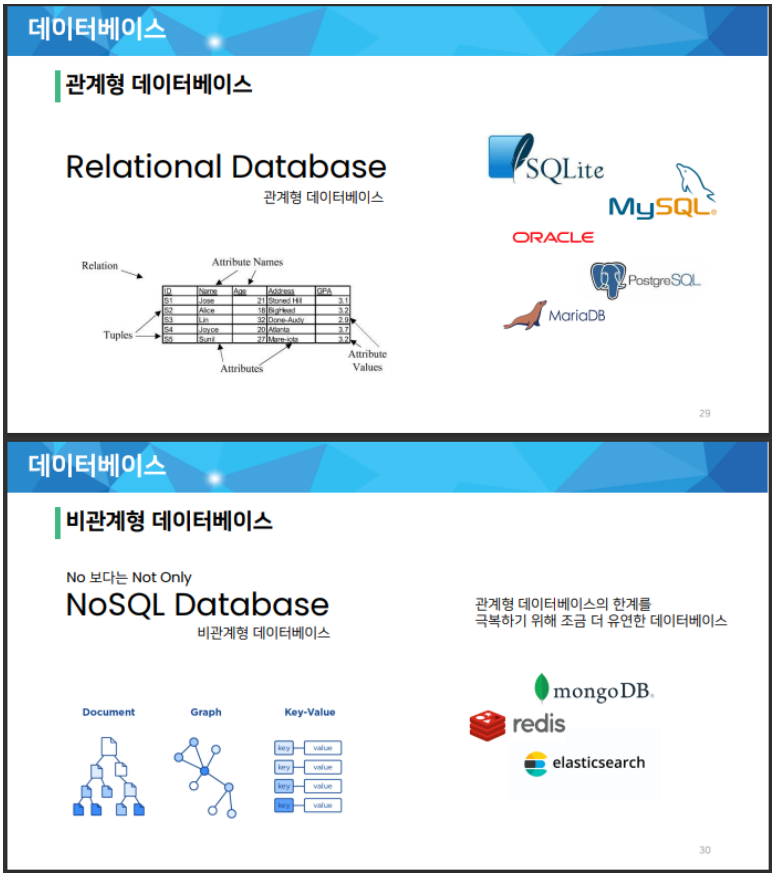
(data base management system)

예를들면 sqlite mysql mariaDB orcle MongoDB redis 등등 종류가 많다.

이중에서 데이터 베이스의 형태에 따라 크게 두개로 나눌 수 있다

관계형 Database. 와. 비관계형 database

SQL vs NOSQL. 로 나누는데



관계형 DB는 우리가 django실습에 봤던 표의 형태로 데이터를 관리하는 것을 말하며

비관계형은 트리 그래프 Key-Value 형태로 비 정형화된 데이터들을 관리하는 것을 말한다.

관계형 DB의 소프트웨어로 sqlite mysql mariaDB orcle등이 있고

비관계형 DBMS로는 mongoDB 또는 redis 등이 있다.

실제로 두 데이터베이스 모두 많이 사용이 되는데

일반적으로 메인 데이터베이스는 관계형으로 사용하고

서브로 NOSQL을 사용하는데 NOSQL을 추가로 사용할 경우는

실시간 사진 또는 채팅 메시지 처리하는 등 복잡하고 빠른 데이터 처리가 필요할 경우에

많이 사용한다.

다시 돌아가서 이러한 데이터 베이스를 조작하기 위해서 우리는

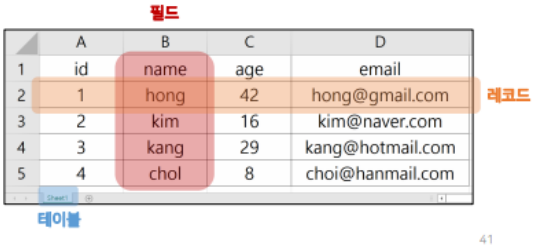
데이터베이스를 조작하는 언어인 SQL을 학습할 것이다.

SQL (Structured Query Language)

구조화된 데이터요청하는 언어

==========

column	datatype
id	INTEGER
name	TEXT
address	TEXT
age	INTEGER



SQL 연습하기 전에 용어 부터 보자

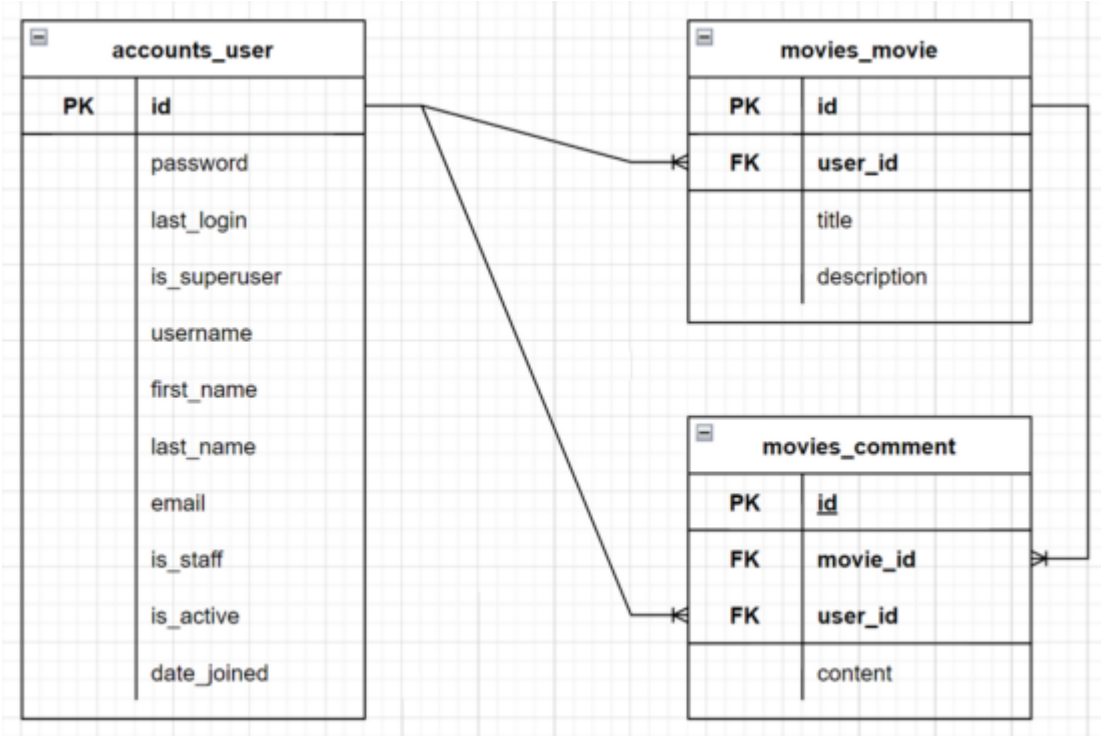
스키마 - 테이블을 어떻게 만들 지에 대한 전반적인 명세서이자 청사진 (왼쪽사진)

테이블(table) - 레코드(col) - 필드(row)

pk - 각 레코드의 고유한 값

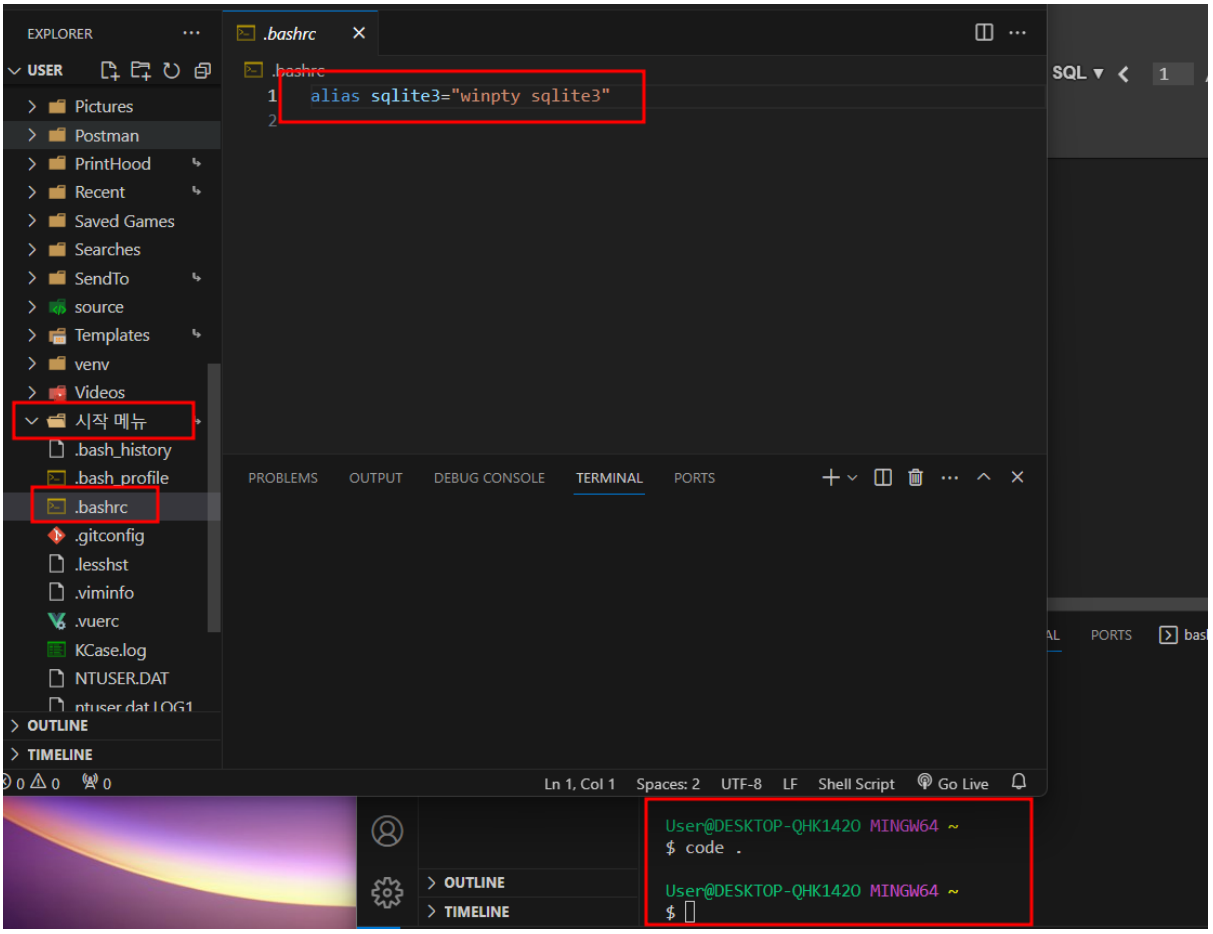
fk - 다른 테이블의 레코드를 식별할 수 있는 키로써

다른 테이블의 레코드를 참조해서 테이블 간의 관계를 만들 때 사용


accounts_user
PK
id
password
last_login
is_superuser
username
first_name
last_name
email
is_staff
is_active
date_joined
movies_movie
PK
id
FK
user_id
title
description
movies_comment
PK
id
FK
movie_id
FK
user_id
content

==========

실습환경 셋팅



git bash는 git 명령어와 유닉스/리눅스 쉘 프로그래밍을 하는 프로그램인데

(쉘 프로그래밍 - 키보드로 입력된 명령을 운영체제로 전달해서 실행하게 되는 프로그램)

윈도우 환경에서 리눅스 쉘 프로그램인 git bash를 통해서 명령어를 작성하면 두 운영체제 (윈도우와 리눅스) 간에 완벽하게 호환이 안되는 경우가 있다. 그럴 때 사용하는 것이 winpty 다.

그 다음에 시스템속성 > 고급 > 환경변수 > 클릭

아래 네모칸 시스템 환경변수에서 path 찾아서 더블클릭

새로만들기 클릭 후 C:₩sqlite 넣은 후 확인 확인 확인을 누르면 끝난다.

vscdoe 터미널 창에 sqlite3 라고 써봐라.

만약에 안되면 chocolately 를 사용해서 환경변수 설정을 하자.

==========

참고로 chocolatey 는 윈도우 환경에서 사용하는 커맨드라인 패키지 매니저 프로그램이다. 프로그램 설치하면 환경변수를 자동으로 셋팅해 주는 소프트웨어 인데 궁금하신 분은 google 검색 해봐라.

먼저 chocolatey 를 사용해 SQLite 를 설치하자.

파워셸을 관리자권한으로 열고, 다음 명령을 입력해서 설치한다.

```
$ choco install sqlite
```

그리고 vscode를 새로 열어서 sqlite3 라고 타이핑 해보자.

==========

SQL 을 살펴보자.

분류	개념	SQL 키워드
DDL - 데이터 정의 언어 (Data Definition Language)	관계형 데이터베이스 구조(테이블, 스키마)를 정의(생성, 수정 및 삭제)하기 위한 명령어	CREATE DROP ALTER
DML - 데이터 조작 언어 (Data Manipulation Language)	데이터를 조작(추가, 조회, 변경, 삭제) 하기 위한 명령어	INSERT SELECT UPDATE DELETE
DCL - 데이터 제어 언어 (Data Control Language)	데이터의 보안, 수행제어, 사용자 권한 부여 등을 정의 하기 위한 명령어	GRANT REVOKE COMMIT ROLLBACK

1. DB테이블의 구조를 관리 (생성 수정 삭제 하기 위함) - create // alter // drop
    a. DDL data definition language (데이터 정의언어)라고 도 하며

2. 데이터를 추가 조회 변경 삭제 - insert select updata delete
    a. DML data manipulation language (데이터 조작언어) 라 한다.

그 외에도

3. 데이터의 보안, 수행제어 ,사용자 권한부여 - commit / rollback // grant // revoke 있지만

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

SQLite는 파일로 데이터를 관리하는 가벼운 database 이며
SQL 언어의 모든 문법과 기능을 지원하지 않는다.

예를 들면 SQL문의 Grant 랑 Revoke와 같이 권한 부여 및 회수는 지원하지 않는다.
(파일로 데이터를 관리하니까 그렇겠쥬?)

자 이제 본격적으로 DDL을 해보자.

CREATE TABLE

mydb.sqlite3
first.sql 이라고 파일 생성하자.

- `.sql` : SQL을 작성하고 실행시키는 파일 (sql문을 적을 연습장)
- `.sqlite3` : SQLite DB

`CREATE TABLE` statement

- Create a new table in the database
- 데이터베이스에 새 테이블을 만듦

CREATE TABLE 실습

- contacts 테이블 생성하기

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

  -

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

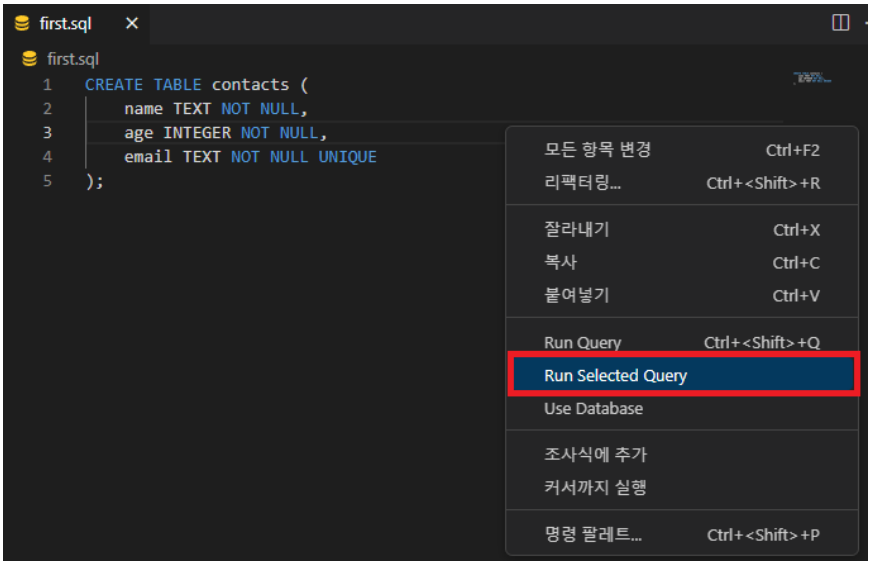
  총 네 개의 컬럼(column) 으로 구성되어있으며, 각각 id, name , age , address 이다.
  필드 다음 나오는 TEXT , INTEGER 는 무엇을 의미하나? 데이터 타입을 의미한다.
  NOT NULL 은 공백 허용 안함이다. NOT NULL 이 있다면 반드시 데이터가 들어가야 한다.

  각 컬럼의 행의 끝은 콤마 , 로 끝나고, 맨 끝 컬럼엔 콤마를 붙여주지 않는다.

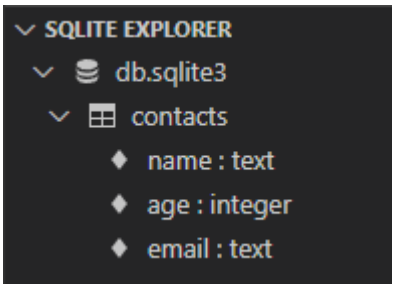
  모든 컬럼을 소괄호 () 로 감싼 다음, 맨 마지막엔 반드시 세미콜론 ; 을 붙여준다.
  이렇게 만들어진 테이블 안에 아직은 실제 데이터가 존재하지 않는다. 그래도 확인해 보자.

- Query 실행하기
  - 실행하고자 하는 명령문에 커서를 두고 마우스 우측 버튼

    → Run Selected Query 선택
  - 명령문을 모두 선택할 필요 없으며, 실행하고자 하는 명령문 안에 커서가 올라가 있으면 가능



- 쿼리 실행 후 테이블 및 스키마 확인



SQLite Data Types

Data Types 종류 (ppt 197page)

Example Typenames From The CREATE TABLE Statement	Resulting Affinity
INT, INTEGER, TINYINT SMALLINT, MEDIUMINT, BIGINT UNSIGNED, BIG INT INT2, INT8	INTEGER
CHARACTER(20), VARCHAR(255), VARYING CHARACTER(255) NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100) TEXT, CLOB	TEXT
BLOB (no datatype specified)	BLOB
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC

83

1. NULL
    - 정보가 없거나 알 수 없음을 의미 (missing information or unknown)
2. INTEGER
    - 정수
3. REAL
    - 실수
4. TEXT
    - 문자 데이터
5. BLOB (Binary Large Object)
    - 입력된 그대로 저장된 데이터 덩어리
    - 예) 이미지 데이터 (~.png) 파일이 있지만 컴퓨터는 이 파일을 010101 이진수로 인코딩 된 데이터 덩어리로 받아들이는데 이 타입을 BLOB 라 한다.

- 뉴메릭은 SQLite에서 지원하지 않음!! SQLite 에서는 INT 나 REAL로 다 표현함

Type Affinity

- 타입 선호도
- 특정 컬럼에 저장된 데이터에 권장되는 타입

- 데이터 타입 작성 시 SQLite의 5가지 데이터 타입이 아닌 다른 데이터 타입을 선언한다면, 내부적으로 각 타입의 지정된 선호도에 따라 5가지 선호도로 인식됨
    1. INTEGER
    2. TEXT
    3. BLOB
    4. REAL
    5. NUMERIC
- 타입 선호도 존재 이유
    - 다른 데이터베이스 엔진 간의 호환성 최대화
    - 정적이고 엄격한 타입을 사용하는 데이터베이스의 SQL 문을 SQLite에서도 작동하도록 하기 위함

[참고]

```
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 가능
한데, Django에서 ORM 을 통해 SQLite 에 보내는 SQL 은 Django ORM이 알아서 적
절히 번역해줘서 날짜 시간 참 거짓 등이 가능 했던 것이다.
원래의 SQLite 에선 bool 과 같은 타입들은 존재하지 않는다.
```

Costraints

💡 **개요**
- 제약 조건
- 입력하는 자료에 대해 제약을 정함
- 제약에 맞지 않다면 입력이 거부됨
- 사용자가 원하는 조건의 데이터만 유지하기 위한 즉, 테이블의 특정 컬럼에 설정하는 제약

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

Constratints 종류

1. NOT NULL
    - 컬럼이 NULL 값을 허용하지 않도록 지정 (값이 비어있으면 안된다!!)
    - 명시를 안하면 기본 default 값으로 NULL 값을 허용함
2. UNIQUE
    - 컬럼의 모든 값이 서로 구별되거나 고유한 값이 되도록 함
    - 예를들어 한번 등록한 email 또 등록 못하게함
3. PRIMARY KEY
    - 테이블에서 행의 고유성을 식별하는데 사용되는 컬럼
    - 각 테이블에는 하나의 기본 키만 있음
    - 암시적으로 NOT NULL 제약 조건이 포함되어 있음
        - 주의 ! INTEGER 타입에서만 사용 가능
4. AUTOINCREMENT
    - INTEGER PRIMARY KEY 다음에 작성해서 해당 row id를 다시 재사용 하지 못하도록 할 때 사용

```
CREATE TABLE table_name (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
);
```

```
## [참고] rowid 특징

- 테이블을 생성할 때마다 rowid라는 암시적 자동 증가 컬럼이 "자동으로 생성"될 것이다.
- 테이블에 새 행을 삽입할 때마다 정수 값을 자동으로 할당한다.
```

```
- 데이터가 최대 값에 도달하고 새 행을 삽입하려고 하면 SQLite는 사용되지 않는 정수를 찾아 사용
- 만약 SQLite가 사용되지 않은 정수를 찾을 수 없으면 SQLITE_FULL 에러가 발생한다.
```

ALTER TABLE

💡 **개요**

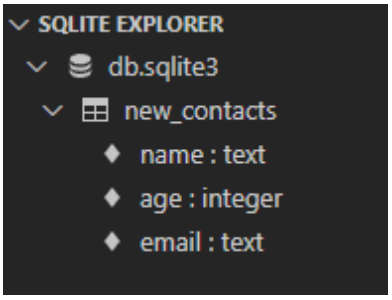
- 기존 테이블의 구조를 수정(변경)

ALTER TABLE statement 예시

ALTER TABLE RENAME

- Rename a table (테이블명 변경)
- 작성 및 결과 확인

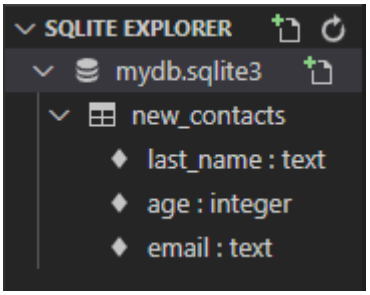
```
ALTER TABLE contacts RENAME TO new_contacts;
```



ALTER TABLE RENAME COLUMN

- Rename a column (컬럼명 변경)
- 작성 및 결과 확인

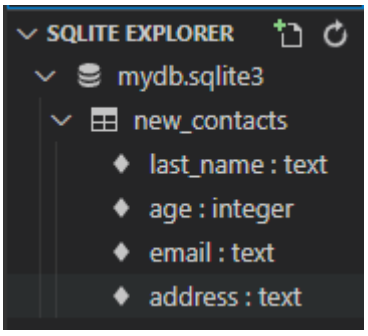
```
ALTER TABLE new_contacts RENAME COLUMN name TO last_name;
```



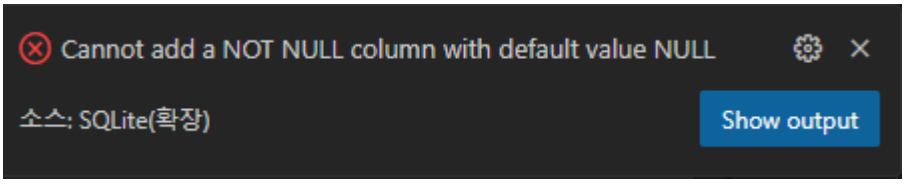
ALTER TABLE ADD COLUMN

- Add a new column to a table (새 컬럼 추가)
- 작성 및 결과 확인

```
ALTER TABLE new_contacts ADD COLUMN address TEXT NOT NULL;
```



- 현재 과정에서는 일어나지 않지만 만약 테이블에 기존 데이터가 있을 경우 다음과 같은 에러가 발생



- 이전에 이미 저장된 데이터들은 새롭게 추가되는 컬럼에 값이 없기 때문에 NULL이 작성되 있을것임
- 그런데 새로 추가되는 컬럼에 NOT NULL 제약조건이 있기 때문에 기본 값 없이는 추가될 수 없다는 에러가 발생한 것
- 다음과 같이 `DEFAULT` 제약 조건을 사용하여 해결할 수 있음

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

- 이렇게 하면 address 컬럼이 추가되면서 기존에 있던 데이터들의 address 컬럼에는 `no address`라는 값이 들어가게 됨

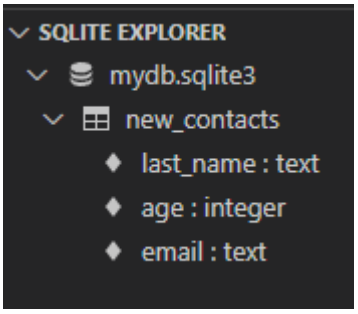
ALTER TABLE DROP COLUMN

- Delete a column (컬럼 삭제)
  - column을 지우는 것은 `sqlite`를 터미널에서 직접 켜서 타이핑 해야 작동됨

```
$ sqlite3 db.sqlite3
```

- 작성 및 결과 확인

```
ALTER TABLE new_contacts DROP COLUMN address;
```



- 단, 삭제하지 못하는 경우가 있음
  - 컬럼이 다른 부분에서 참조되는 경우
    - FOREIGN KEY(외래 키) 로 사용되는 경우
  - PRIMARY KEY 인 경우
  - UNIQUE 제약 조건이 있는 경우 불가능!

```
Cannot drop UNIQUE column: "email"   이런식으로 버그 메시지 뜸
```

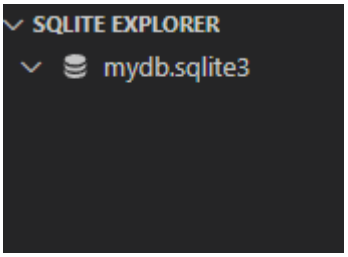
DROP TABLE

- Remove a table from the database
- 만약에 존재하지 않는 테이블을 제거하려면 SQLite에서 오류가 발생

```
no such table: table_name
```

- 작성 및 결과 확인

```
DROP TABLE new_contacts;
```



DROP TABLE 특징

- 한 번에 하나의 테이블만 삭제할 수 있음
- 여러 테이블을 제거하려면 여러 DROP TABLE 문을 실행해야 함
- DROP TABLE 문은 실행 취소하거나 복구할 수 없음
    - 따라서 각별히 주의하여 실행하여야 함!

📌 요약

- DDL을 사용하여 데이터베이스 테이블 구조를 관리하는 방법

```
-- contacts 테이블 생성하기
CREATE TABLE contacts (
    name TEXT NOT NULL,
    age INTEGER NOT NULL,
    email TEXT NOT NULL UNIQUE
);

-- 테이블 이름 변경하기
ALTER TABLE contacts RENAME TO new_contacts;
```

```
-- 컬럼 이름 변경하기
ALTER TABLE new_contacts RENAME COLUMN name TO last_name;

-- 새 컬럼 추가하기
ALTER TABLE new_contacts ADD COLUMN address TEXT NOT NULL;
-- 새 컬럼 추가시 기존데이터 있을 경우 발생하는 오류 방지하기
ALTER TABLE new_contacts
ADD COLUMN address TEXT NOT NULL DEFAULT 'no address';

-- 컬럼 삭제하기
ALTER TABLE new_contacts DROP COLUMN address;

-- 테이블 삭제하기
-- 제거하면 되돌릴 수 없으므로 조심하자!
DROP TABLE contacts;
```

DML 을 살펴보자.

분류	개념	SQL 키워드
DDL - 데이터 정의 언어 (Data Definition Language)	관계형 데이터베이스 구조(테이블, 스키마)를 정의(생성, 수정 및 삭제)하기 위한 명령어	CREATE DROP ALTER
DML - 데이터 조작 언어 (Data Manipulation Language)	데이터를 조작(추가, 조회, 변경, 삭제) 하기 위한 명령어	INSERT SELECT UPDATE DELETE
DCL - 데이터 제어 언어 (Data Control Language)	데이터의 보안, 수행제어, 사용자 권한 부여 등을 정의 하기 위한 명령어	GRANT REVOKE COMMIT ROLLBACK

💡 **DML을 통해 데이터를 조작하기 (CRUD)**
INSERT, SELECT, UPDATE, DELETE

---

command-line program - sqlite3

CSV 파일을 SQLite 테이블로 가져오기

- 실습을 위해서 CSV 파일을 테이블로 가져오기를 할 것이다.
-

1. DML.sql 라는 이름의 파일을 하나 생성하자. 그리고 mydb.sqlite3 파일이 없다면 만들자.
2. `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 존재 해야 한다.

다음과 같은 쿼리문을 `test.sql` 에 작성해 실행한다.

1. 테이블 생성하기

```
CREATE TABLE users (
    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 반드시 저장을 눌러라.
그리고 run selected query도 행하라!
```

2. 데이터베이스 파일 열기

```
$ sqlite3 mydb.sqlite3
```

3. 모드(.mode)를 csv로 설정

```
sqlite> .mode csv
```

4. .import 명령어를 사용하여 csv 데이터를 테이블로 가져오기

```
sqlite> .import users.csv users
```

5. Open database로 import 된 데이터 확인하기

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

Simple query

💡 SELECT 문을 사용하여 간단하게 단일 테이블에서 데이터 조회하기

SELECT statement

- 특정 테이블에서 데이터를 조회하기 위해 사용하는 데 문법은 다음과 같다.

```
SELECT column1, column2 FROM table_name;
```

- 문법 규칙
  1. SELECT 절에서 컬럼 또는 쉼표로 구분된 컬럼 목록을 지정
  2. FROM 절(clause)에서 데이터를 가져올 테이블을 지정

- SELECT 문은 SQLite에서 가장 복잡한 문법이다. SELECT문법은 다양한 절과 함께 응용 할 수 있다.
- 다양한 절과 함께 사용할 수 있다.
    - ORDER BY, DISTINCT, WHERE, LIMIT, LIKE, GROUP BY

SELECT 실습

- 테이블 명에 있는 모든(`*`) 컬럼 조회 (모든 데이터 조회)
- `limit` : 보여줄 행의 개수 설정하기

```
SELECT * FROM users limit 20;
```

- 이름과 나이 조회하기

```
SELECT first_name, age FROM users 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

- rowid, 이름 조회하기

```
SELECT rowid, first_name FROM users;
```

Sorting rows

💡 ORDER BY 절을 사용하여 쿼리의 결과 정렬하기

ORDER BY clause

```
SELECT select_list FROM table_name
ORDER BY column_1 ASC, column_2 DESC;
```

- SELECT 문에 ORDER BY 절을 추가하여 결과를 정렬
- ORDER BY 절은 FROM 절 뒤에 위치한다.
- 하나 이상의 컬럼을 기준으로 결과를 오름차순, 내림차순으로 정렬할 수 있음
- 이를 위해 ORDER BY 절 다음에 ‘ASC’ 또는 ‘DESC’ 키워드 사용 가능하다.
  - ASC : 오름차순 (기본 값)
  - DESC : 내림차순

ORDER BY 실습

- 이름과 나이를 나이가 어린 순서대로 조회하기

```
SELECT first_name, age FROM users ORDER BY age ASC;
SELECT first_name, age FROM users ORDER BY age;
```

- 이름과 나이를 나이가 많은 순서대로 조회하기

```
SELECT first_name, age FROM users ORDER BY age DESC;
```

- 이름, 나이, 계좌 잔고를 나이가 어린 순으로, 만약 같은 나이라면 계좌 잔고가 많은 순으로 정렬해서 조회하기

```
SELECT first_name, age, balance FROM users
ORDER BY age, balance DESC;
```

- ORDER BY 절은 하나 이상의 컬럼을 정렬할 경우 첫 번째 열을 사용하여 행을 정렬하고, 그 다음 두번째 컬럼을 사용하여 정렬되어 있는 행을 정렬하는 방식
- 즉, 먼저 age를 기준으로 먼저 오름차순 정렬하고, 이 결과를 balance를 기준으로 내림차순으로 정렬한 것

[참고] Sorting NULLs

- NULL의 정렬 방식
- 정렬과 관련하여 SQLite는 NULL을 다른 값보다 작은 것으로 간주
- 즉, ASC를 사용하는 경우 결과의 시작 부분에 NULL이 표시되고, DESC를 사용하는 경우 결과의 끝에 NULL이 표시됨

Filtering data

💡 **개요**
- 데이터를 필터링 하여 중복 제거, 조건 설정 등 쿼리를 제어하기
- Clause : SELECT DISTINCT, WHERE, LIMIT
- Operator : LIKE, IN BETWEEN

SELECT DISTINCT clause

```
SELECT DISTINCT select_list FROM table_name;
```

- 조회 결과에서 중복된 행을 제거
- DISTINCT 절은 SELECT 에서 선택적으로 사용할 수 있는 절
- 문법 규칙
  1. DISTINCT 절은 SELECT 키워드 바로 뒤에 나타나야 함
  2. DISTINCT 키워드 뒤에 컬럼 또는 컬럼 목록 작성

SELECT DISTINCT 실습

- 중복 없이 모든 지역 조회하기

```
SELECT DISTINCT country FROM users;
```

- 지역 순으로 오름차순 정렬하여 중복 없이 모든 지역 조회하기

```
SELECT DISTINCT country FROM users ORDER BY country;
```

- 이름과 지역이 중복 없이 모든 이름과 지역 조회하기

```
SELECT DISTINCT first_name, country FROM users;
```

- 이름과 지역 중복 없이 지역 순으로 오름차순 정렬하여 모든 이름과 지역 조회하기

```
SELECT DISTINCT first_name, country
FROM users
ORDER BY country;
```

[참고] NULL with DISTINCT

- SQLite는 NULL 값을 중복으로 간주
- NULL 값이 있는 컬럼에 DISTINCT 절을 사용하면 SQLite는 NULL 값의 한 행을 유지

WHERE clause

```
SELECT column_list FROM table_name
WHERE search_condition;
```

- 조회 시 특정 검색 조건을 지정
- WHERE 절은 SELECT 문에서 선택적으로 사용할 수 있는 절 이다.
    - SELECT 문 외에도 뒤에 나올 UPDATE 및 DELETE 문에서 WHERE 절을 사용할 수 있다.
- 작성 위치는 FROM 절 뒤에 작성한다.

WHERE의 검색 조건 작성 형식

```
left_expression COMPARISON_OPERATOR right_expression
```

```
WHERE column_1 = 10

WHERE column_2 LIKE 'Ko%'

WHERE column_3 in (1,2)

WHERE column_4 BETWEEN 10 AND 20
```

SQLite comparison operators (비교연산자)

- 두 표현식이 동일한지 테스트
  - =
  - <> or ≠
  - <
  - >
  - <=
  - >=

SQLite logical operators (논리연산자)

- 일부 표현식의 truth를 테스트할 수 있음
- 1, 0 또는 NULL 값을 반환
- SQLite는 Boolean 데이터 타입을 제공하지 않으므로 1은 TRUE를 의미하고 0은 FALSE를 의미
- ALL, AND, ANY, BETWEEN, IN, LIKE, NOT, OR 등

WHERE 실습해보자.

- 나이가 30살 이상인 사람들의 이름, 나이, 계좌 잔고 조회하기

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

- 나이가 30살 이상이고 계좌 잔고가 50만원 초과인 사람들의 이름, 나이, 계좌 잔고 조회하기

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

AND 대신 OR을 적어 주었다면? 나이가 30 이상이거나 잔고가 50 초과인 사람을 뜻한다.

LIKE operator

- Qurey data based on pattern matching
- 패턴 일치를 기반으로 데이터를 조회
- SELECT, DELETE, UPDATE 문의 WHERE 절에서 사용
- 기본적으로 대소문자 구분하지 않음
- SQLite는 패턴 구성을 위한 두 개의 와일드카드(wildcards)를 제공
  1. % (percent)
     - 0 개 이상의 문자가 올 수 있음을 의미
  2. _ (underscore)
     - 단일(1개) 문자가 있음을 의미

% wildcard 예시

- '영%' : 영으로 시작하는 모든 문자열과 일치 (영, 영미, 영미리 등)
- '%도' : 도로 끝나는 모든 문자열과 일치 (도, 수도, 경기도 등)
- '%강원%' : 강원을 포함하는 모든 문자열과 일치 (강원, 강원도, 강원도에 살아요 등)

_ wildcard 예시

- '영_' : 영으로 시작하고 총 2자리인 문자열과 일치 (영미, 영수, 영호 등)
- '_도': 도로 끝나고 총 2자리인 문자열과 일치 (수도, 과도 등)

wildcard 종합 예시

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 이상)

LIKE 실습

- 이름에 '호'가 포함되는 사람들의 이름과 성 조회하기

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

- 이름이 '준'으로 끝나는 사람들의 이름 조회하기

```
SELECT first_name FROM users
WHERE first_name LIKE '%준';
```

- 서울 지역 전화번호를 가진 사람들의 이름과 전화번호 조회하기

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

- 나이가 20대인 사람들의 이름과 나이 조회하기

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

- 전화번호 중간 4자리가 51로 시작하는 사람들의 이름과 전화번호 조회하기

```
SELECT first_name, phone FROM users
WHERE phone LIKE '%-51__-%';
```

IN operator

- Determine whether a value matches any value in a list of values
- 값이 값 목록 결과에 있는 값과 일치하는지 확인
- 표현식이 값 목록의 값과 일치하는지 여부에 따라 true 또는 false 반환
- IN 연산자의 결과를 부정하려면 `NOT IN` 연산자 사용

IN 실습

- 경기도 혹은 강원도에 사는 사람들의 이름과 지역 조회하기

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

  - IN 연산자 대신 OR 연산자를 사용하여 동일한 결과를 반환할 수 있다.

```
SELECT first_name, country FROM users
WHERE country == '경기도' or country ='강원도';
```

- 경기도 혹은 강원도에 살지 않는 사람들의 이름과 지역 조회하기

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

BETWEEN operator

```
test_expression BETWEEN low_expression AND high_expression
```

- 값이 범위 안에 있는지 테스트
- 값이 지정된 범위에 있으면 true를 반환
- SELECT, DELETE, 및 UPDATE 문의 WHERE 절에서 사용할 수 있음
- BETWEEN 연산자의 결과를 부정하려면 `NOT BETWEEN` 연산자 사용

BETWEEN 실습

- 나이가 20살 이상, 30살 이하인 사람들의 이름과 나이 조회하기

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

  - AND 연산자를 사용하여 동일한 결과 반환할 수 있음

```
SELECT first_name, age FROM users
WHERE age >= 20 AND age <= 30;
```

- 나이가 20살 이상, 30살 이하가 아닌 사람들의 이름과 나이 조회하기

```
SELECT first_name, age FROM users
WHERE age NOT BETWEEN 20 AND 30;
```

  - OR 연산자를 사용하여 동일한 결과 반환할 수 있음

```
SELECT first_name, age FROM users
WHERE age < 20 OR age > 30;
```

LIMIT clause

```
SELECT column_list FROM table_name LIMIT row_count;
```

- Constrain the number of rows returned by a query
- 쿼리에서 반환되는 행 수를 제한
- SELECT 문에서 선택적으로 사용할 수 있는 절
- row_count는 반환되는 행 수를 지정하는 양의 정수를 의미

LIMIT 실습

- 첫 번째부터 열 번째 데이터까지 rowid와 이름 조회하기

```
SELECT rowid, first_name FROM users LIMIT 10;
```

- 계좌 잔고가 가장 많은 10명의 이름과 계좌 잔고 조회하기
  - ORDER BY 절과 함께 사용하여 지정된 순서로 여러 행을 가져올 수도 있음

    - LIMIT 절에 지정된 행 수를 가져오기 전에 결과를 정렬하기 때문

```
SELECT first_name, balance FROM users
ORDER BY balance DESC LIMIT 10;
```

- 나이가 가장 어린 5명의 이름과 나이 조회하기

```
SELECT first_name, age FROM users
ORDER BY age LIMIT 5;
```

OFFSET keyword

- LIMIT 절을 사용하면 첫 번째 데이터부터 지정한 수 만큼의 데이터를 받아올 수 있지만, OFFSET과 함께 사용하면 특정 지정된 위치에서부터 데이터를 조회할 수 있음
- 11번째부터 20번째 데이터의 rowid와 이름 조회하기

```
SELECT rowid, first_name FROM users
LIMIT 10 OFFSET 10;
```

Grouping Data

Aggregate function

- 집계함수
- 값 집합의 최대값, 최소값, 평균, 함계 및 개수를 계산
- 값 집합에 대한 계산을 수행하고 단일 값을 반환
    - 여러 행으로부터 하나의 결과 값을 반환하는 함수
- SELECT 문의 GROUP BY 절과 함께 종종 사용됨
- 제공하는 함수 목록
    - AVG(), COUNT(), MAX(), MIN(), SUM()
- AVG(), MAX(), MIN(), SUM()는 숫자를 기준으로 계산이 되어져야 하기 때문에 반드시 컬럼의 데이터 타입이 숫자(INTEGER)일 때만 사용 가능

Aggregate function 실습

- users 테이블의 전체 행 수 조회하기

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

- 전체 유저의 평균 balance 조회하기

```
SELECT avg(balance) From users;
```

GROUP BY clause

```
SELECT column_1, aggregate_function(column_2) FROM table_name
GROUP BY column_1, column_2;
```

- Make a set of summary rows from a set of rows
- 특정 그룹으로 묶인 결과를 생성
- 선택된 컬럼 값을 기준으로 데이터(행)들의 공통 값을 묶어서 결과로 나타냄
- SELECT 문에서 선택적으로 사용 가능한 절
- SELECT 문의 FROM 절 뒤에 작성
  - WHERE 절이 포함된 경우 WHERE 절 뒤에 작성해야 함
- 각 그룹에 대해 MIN, MAX, SUM, COUNT 또는 AVG와 같은 집계함수(aggregate function)를 적용하여 각 그룹에 대한 추가적인 정보 제공 가능

Aggregate function 실습

- 지역별 평균 balance 구하기

```
SELECT country, avg(balance) FROM users
GROUP BY country;

-- balance 기준 오름차순 정렬하여 조회하기
SELECT country, avg(balance) FROM users
GROUP BY country ORDER BY avg(balance) DESC;
```

- 나이가 30살 이상인 사람들의 평균 나이 구하기

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

GROUP BY 실습

- 각 지역별로 몇 명씩 살고 있는지 조회하기
  - 지역별로 그룹을 나누어야 한다.

    ```
    SELECT country FROM users GROUP BY country;
    ```

  - 그룹별로 포함되는 데이터 수를 구함

    → Aggregation Function 의 COUNT 사용

    ```
    SELECT country, COUNT(*) FROM users GROUP BY country;
    ```

  - 참고 사항
    - 이전 쿼리에서 COUNT(), COUNT(age), COUNT(last_name) 등 어떤 컬럼을 넣어도 결과는 같음
    - 현재 쿼리에서는 그룹화된 country를 기준으로 카운트하는 것이기 때문에 어떤 컬럼을 카운트해도 전체 개수는 동일하기 때문
- 각 성씨가 몇 명씩 있는지 조회하기

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

-- AS 키워드를 사용해 컬럼명 변경가능
SELECT last_name, COUNT(*) AS number_of_name FROM users
GROUP BY last_name;
```

- 인원이 가장 많은 성씨 순으로 조회하기

```
SELECT last_name, COUNT(*) FROM users
GROUP BY last_name ORDER BY COUNT(*) DESC;
```

- 각 지역별 평균 나이 조회하기

```
SELECT country, AVG(age) FROM users
GROUP BY country;
```

Changing data

- 데이터를 삽입, 수정, 삭제하기 : INSERT, UPDATE, DELETE
- 사전 준비 : 새 테이블 생성

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

INSERT statement

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2, ...);
```

새 행을 테이블에 삽입

1. 먼저 INSERT INTO 키워드 뒤에 데이터를 삽입할 테이블의 이름을 지정
2. 테이블 이름 뒤에 쉼표로 구분된 컬럼 목록을 추가
    - 컬럼 목록은 선택사항이지만 컬럼 목록을 포함하는 것이 권장됨
3. VALUES 키워드 뒤에 쉼표로 구분된 값 목록 추가
    - 만일 컬럼 목록을 생략하는 경우 값 목록의 모든 컬럼에 대한 값을 지정해야 함
    - 값 목록의 값 개수는 컬럼 목록의 컬럼 개수와 같아야 함

INSERT 실습

- 단일 행 삽입하기

```
INSERT INTO classmates (name, age, address)
VALUES ('홍길동', 23, '서울');
```

```
INSERT INTO classmates
VALUES ('홍길동', 23, '서울');
```

- 여러 행 삽입하기

```
INSERT INTO classmates
VALUES
    ('김철수', 30, '경기'),
    ('이영미', 31, '강원'),
    ('박진성', 26, '전라'),
    ('최지수', 12, '충청'),
    ('정요한', 28, '경상');
```

UPDATE statement

```
UPDATE table_name
SET column_1 = new_value_1,
    column_2 = new_value_2
WHERE
    search_condition;
```

1. UPDATE 절 이후에 업데이트할 테이블을 지정
2. SET 절에서 테이블의 각 컬럼에 대해 새 값을 설정
3. WHERE 절의 조건을 사용하여 업데이트할 행을 지정
    - WHERE 절은 선택 사항
    - 생략하면 UPDATE 문은 테이블의 모든 행에 있는 데이터를 업데이트 함
4. 선택적으로 ORDER BY 및 LIMIT 절을 사용하여 업데이트 할 행 수를 지정할 수도 있음

UPDATE 실습

- 2번 데이터의 이름을 '김철수한무두루미', 주소를 '제주도'로 수정하기

```
UPDATE classmates
SET name = '김철수한무두루미',
    address = '제주도'
WHERE rowid = 2;
```

DELETE statement

```
DELETE FROM table_name
WHERE search_condition;
```

- 테이블에서 행을 제거
- 테이블의 한 행, 여러 행 및 모든 행을 삭제할 수 있음

1. DELETE FROM 키워드 뒤에 행을 제거하려는 테이블의 이름을 지정
2. WHERE 절에 검색 조건을 추가하여 제거할 행을 식별
    - WHERE 절은 선택사항이며, 생략하면 DELETE 문은 테이블의 모든 행을 삭제
3. 선택적으로 ORDER BY 및 LIMIT 절을 사용하여 삭제할 행 수를 지정할 수도 있음

DELETE 실습

- 5번 데이터 삭제하기

```
DELETE FROM classmates WHERE rowid = 5;

-- 삭제된 것 확인
SELECT rowid, * FROM classmates;
```

- 이름에 '영'이 포함되는 데이터 삭제하기

```
DELETE FROM classmates WHERE name LIKE '%영%';
```

- 테이블의 모든 데이터 삭제하기

```
DELETE FROM classmates;
```

📍 요약

```
-- 모든 데이터 조회하기
SELECT * FROM users;

-- 이름, 나이 조회하기
SELECT first_name, age FROM users;

-- rowid, first_name 조회하기
SELECT rowid, first_name FROM users;
```

```sql
-- 나이가 어린 순서대로 이름과 나이 조회하기
SELECT first_name, age FROM users ORDER BY age ASC;
SELECT first_name, age FROM users ORDER BY age;

-- 나이가 많은 순서대로 이름과 나이 조회하기
SELECT first_name, age FROM users ORDER BY age DESC;

-- 이름, 나이, 계좌 잔고를 나이가 어린 순으로,
-- 만약 같은 나이라면 계좌 잔고가 많은 순으로 정렬해서 조회하기
SELECT first_name, age, balance FROM users
ORDER BY age, balance DESC;
```

```sql
-- 중복 없이 모든 지역 조회하기
SELECT DISTINCT country FROM users;

-- 지역 순으로 오름차순 정렬하여 중복 없이 모든 지역 조회하기
SELECT DISTINCT country FROM users ORDER BY country;

-- 이름과 지역이 중복 없이 모든 이름과 지역 조회하기
SELECT DISTINCT first_name, country FROM users;

-- 이름과 지역 중복 없이 지역 순으로 오름차순 정렬하여 모든 이름과 지역 조회하기
SELECT DISTINCT first_name, country
FROM users
ORDER BY country;

-- 나이가 30살 이상인 사람들의 이름, 나이, 계좌 잔고 조회하기
SELECT first_name, age, balance FROM users
WHERE age >= 30;

-- 나이가 30살 이상이고 계좌 잔고가 50만원 초과인 사람들의 이름, 나이, 계좌 잔고 조회하기
SELECT first_name, age, balance FROM users
WHERE age >= 30 AND balance > 500000;

-- 이름에 '호'가 포함되는 사람들의 이름과 성 조회하기
SELECT first_name, last_name FROM users
WHERE first_name LIKE '%호%';

-- 이름이 ‘준’으로 끝나는 사람들의 이름 조회하기
SELECT first_name FROM users
WHERE first_name LIKE '%준';

-- 서울 지역 전화번호를 가진 사람들의 이름과 전화번호 조회하기
SELECT first_name, phone FROM users
WHERE phone LIKE '02-%';

-- 나이가 20대인 사람들의 이름과 나이 조회하기
SELECT first_name, age FROM users
WHERE age LIKE '2_';

-- 전화번호 중간 4자리가 51로 시작하는 사람들의 이름과 전화번호 조회하기
SELECT first_name, phone FROM users
```

```
WHERE phone LIKE '%-51__-%';

-- 경기도 혹은 강원도에 사는 사람들의 이름과 지역 조회하기
SELECT first_name, country FROM users
WHERE country IN ('경기도', '강원도');

SELECT first_name, country FROM users
WHERE country == '경기도' or country ='강원도';

-- 경기도 혹은 강원도에 살지 않는 사람들의 이름과 지역 조회하기
SELECT first_name, country FROM users
WHERE country NOT IN ('경기도', '강원도');

-- 나이가 20살 이상, 30살 이하인 사람들의 이름과 나이 조회하기
SELECT first_name, age FROM users
WHERE age BETWEEN 20 AND 30;

SELECT first_name, age FROM users
WHERE age >= 20 AND age <= 30;

-- 나이가 20살 이상, 30살 이하가 아닌 사람들의 이름과 나이 조회하기
SELECT first_name, age FROM users
WHERE age NOT BETWEEN 20 AND 30;

-- 첫 번째부터 열 번째 데이터까지 rowid와 이름 조회하기
SELECT rowid, first_name FROM users LIMIT 10;

-- 계좌 잔고가 가장 많은 10명의 이름과 계좌 잔고 조회하기
SELECT first_name, balance FROM users
ORDER BY balance DESC LIMIT 10;

-- 나이가 가장 어린 5명의 이름과 나이 조회하기
SELECT first_name, age FROM users
ORDER BY age LIMIT 5;

-- 11번째부터 20번째 데이터의 rowid와 이름 조회하기
SELECT rowid, first_name FROM users
LIMIT 10 OFFSET 10;
```

```
-- users 테이블의 전체 행 수 조회하기
SELECT COUNT(*) FROM users;

-- 전체 유저의 평균 balance 조회하기
SELECT avg(balance) From users;

-- 지역별 평균 balance 구하기
SELECT country, avg(balance) FROM users
GROUP BY country;
-- 지역별 평균 balance를 평균 balance 기준 오름차순 정렬하여 조회하기
SELECT country, avg(balance) FROM users
GROUP BY country ORDER BY avg(balance) DESC;

-- 나이가 30살 이상인 사람들의 평균 나이 구하기
```

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

-- 각 지역별로 몇 명씩 살고 있는지 조회하기
SELECT country, COUNT(*) FROM users GROUP BY country;

-- 각 성씨가 몇 명씩 있는지 조회하기
SELECT last_name, COUNT(*) FROM users
GROUP BY last_name;
-- AS 키워드를 사용해 컬럼명 변경가능
SELECT last_name, COUNT(*) AS number_of_name FROM users
GROUP BY last_name;

-- 인원이 가장 많은 성씨 순으로 조회하기
SELECT last_name, COUNT(*) FROM users
GROUP BY last_name ORDER BY COUNT(*) DESC;

-- 각 지역별 평균 나이 조회하기
SELECT country, AVG(age) FROM users
GROUP BY country;
```

```
-- 단일 행 삽입하기
INSERT INTO classmates (name, age, address)
VALUES ('홍길동', 23, '서울');

INSERT INTO classmates
VALUES ('홍길동', 23, '서울');

-- 여러 행 삽입하기
INSERT INTO classmates
VALUES
    ('김철수', 30, '경기'),
    ('이영미', 31, '강원'),
    ('박진성', 26, '전라'),
    ('최지수', 12, '충청'),
    ('정요한', 28, '경상');

-- 2번 데이터의 이름을 '김철수한무두루미', 주소를 '제주도'로 수정하기
UPDATE classmates
SET name = '김철수한무두루미',
    address = '제주도'
WHERE rowid = 2;

-- 5번 데이터 삭제하기 (지수 삭제 ㅜㅜ)
DELETE FROM classmates WHERE rowid = 5;
-- 삭제된 것 확인
SELECT rowid, * FROM classmates;

-- 이름에 ‘영’이 포함되는 데이터 삭제하기
DELETE FROM classmates WHERE name LIKE '%영%';

-- 테이블의 모든 데이터 삭제하기
DELETE FROM classmates;
```