

Database SQL기초

데이터베이스(DB)

데이터베이스는 체계화된 데이터의 모임이다.

여러 사람이 공유하고 사용할 목적으로 통합 관리되는 정보의 집합이다.
논리적으로 연관된 하나 이상의 자료의 모음으로 그 내용을 고도로 구조화함으로써 검색과
갱신의 효율화를 꾀한 것이다.

즉, 몇 개의 자료 파일을 조직적으로 통합하여 자료 항목의 중복을 없애고
자료를 구조화하여 기억시켜 놓은 자료의 집합체라고 할 수 있다.

출처: 위키피디아

RDBMS (관계형데이터베이스 관리 시스템)

관계형 모델을 기반으로하는 데이터베이스 관리시스템이다.

아래는 대표적인 오픈소스 RDBMS(MySQL, SQLite, PostgreSQL)과 ORACLE, MS SQL이다.


















SQLite는 서버가 아닌 응용 프로그램에 넣어 사용하는 비교적 가벼운 데이터베이스이다.

구글 안드로이드 운영체제에 기본적으로 탑재된 데이터베이스이며,

임베디드 소프트웨어에도 많이 활용이 되고 있다.

로컬에서 간단한 DB 구성을 할 수 있으며, 오픈소스 프로젝트이기 때문에 자유롭게 사용할 수 있다.

| Rank | | | DBMS | Database Model | Score | | |
|----------|----------|---|--|-----------------|----------|----------|----------|
| Jan 2019 | Dec 2018 | Jan 2018 | | | Jan 2019 | Dec 2018 | Jan 2018 |
| 1. | 1. | 1. | Oracle  | Relational DBMS | 1268.84 | -14.39 | -73.11 |
| 2. | 2. | 2. | MySQL  | Relational DBMS | 1154.27 | -6.98 | -145.44 |
| 3. | 3. | 3. | Microsoft SQL Server  | Relational DBMS | 1040.26 | -0.08 | -107.81 |
| 4. | 4. | 4. | PostgreSQL  | Relational DBMS | 466.11 | +5.48 | +79.93 |
| 5. | 5. | 5. | MongoDB  | Document store | 387.18 | +8.57 | +56.24 |
| 6. | 6. | 6. | IBM Db2  | Relational DBMS | 179.85 | -0.90 | -10.43 |
| 7. | 7. |  9. | Redis  | Key-value store | 149.01 | +2.19 | +25.88 |
| 8. | 8. |  10. | Elasticsearch  | Search engine | 143.44 | -1.26 | +20.89 |
| 9. | 9. |  7. | Microsoft Access | Relational DBMS | 141.62 | +2.10 | +14.92 |
| 10. | 10. |  11. | SQLite  | Relational DBMS | 126.80 | +3.78 | +12.54 |

기본 용어 정리

스키마(scheme)

데이터베이스에서 자료의 구조, 표현방법, 관계등을 정의한 구조.

| column | datatype |
|--------|----------|
| id | INT |
| age | INT |
| phone | TEXT |
| email | TEXT |

스키마(scheme)

데이터베이스의 구조와 제약 조건에 관련한 전반적인 명세를 기술한 것.

| column | datatype |
|--------|----------|
| id | INT |
| age | INT |
| phone | TEXT |
| email | TEXT |

| | A | B | C | D | E |
|---|------|---------|-----|---------------|------------------|
| 1 | id | name | age | phone | email |
| 2 | 1 | hong | 42 | 010-1234-1234 | hong@gmail.com |
| 3 | 2 | kim | 16 | 010-1234-5678 | kim@naver.com |
| 4 | 3 | kang | 29 | 010-1111-2222 | kang@hanmail.net |
| 5 | 4 | choi | 8 | 010-3333-4444 | choi@hotmail.com |
| 6 | | | | | |
| 7 | | | | | |
| | user | product | + | | |

테이블(table)

열(Column)

각 열에는 고유한 데이터 형식이 지정된다.
INTEGER TEXT NULL 등

| | A | B | C | D | E |
|---|------|---------|-----|---------------|------------------|
| 1 | id | name | age | phone | email |
| 2 | 1 | hong | 42 | 010-1234-1234 | hong@gmail.com |
| 3 | 2 | kim | 16 | 010-1234-5678 | kim@naver.com |
| 4 | 3 | kang | 29 | 010-1111-2222 | kang@hanmail.net |
| 5 | 4 | choi | 8 | 010-3333-4444 | choi@hotmail.com |
| 6 | | | | | |
| - | | | | | |
| | user | product | + | | |

테이블(table)

행(row), 레코드

테이블의 데이터는 행에 저장된다.
즉, user 테이블에 4명의 고객정보가 저장되어 있으며,
행은 4개가 존재한다.

열(Column)

각 열에는 고유한 데이터 형식이 지정된다.
INTEGER TEXT NULL 등

| | A | B | C | D | E |
|---|------|---------|-----|---------------|------------------|
| 1 | id | name | age | phone | email |
| 2 | 1 | hong | 42 | 010-1234-1234 | hong@gmail.com |
| 3 | 2 | kim | 16 | 010-1234-5678 | kim@naver.com |
| 4 | 3 | kang | 29 | 010-1111-2222 | kang@hanmail.net |
| 5 | 4 | choi | 8 | 010-3333-4444 | choi@hotmail.com |
| 6 | | | | | |
| - | | | | | |
| | user | product | + | | |

테이블(table)

PK(기본키)

각 행(레코드)의 고유값으로 Primary Key로 불린다.
반드시 설정하여야하며, 데이터베이스 관리 및 관계 설정시
주요하게 활용된다.

열(Column)

각 열에는 고유한 데이터 형식이 지정된다.
INTEGER TEXT NULL 등

행(row), 레코드

테이블의 데이터는 행에 저장된다.
즉, user 테이블에 4명의 고객정보가 저장되어 있으며,
행은 4개가 존재한다.

| | A | B | C | D | E |
|---|------|---------|-----|---------------|------------------|
| 1 | id | name | age | phone | email |
| 2 | 1 | hong | 42 | 010-1234-1234 | hong@gmail.com |
| 3 | 2 | kim | 16 | 010-1234-5678 | kim@naver.com |
| 4 | 3 | kang | 29 | 010-1111-2222 | kang@hanmail.net |
| 5 | 4 | choi | 8 | 010-3333-4444 | choi@hotmail.com |
| 6 | | | | | |
| - | | | | | |
| | user | product | + | | |

테이블(table)

1. SQL 개념

SQL

SQL(Structured Query Language)는
관계형 데이터베이스 관리시스템(RDBMS)의 데이터를 관리하기 위해 설계된
특수 목적의 프로그래밍 언어이다.

관계형 데이터베이스 관리 시스템에서 자료의 검색과 관리
데이터베이스 스키마 생성과 수정, 데이터베이스 객체 접근 조정 관리를 위해 고안되었다.

출처: 위키피디아

SQL

SQL 문법은 다음과 같이 세가지 종류로 구분될 수 있다.

| | 개념 | 예시 |
|---|--|---------------------------------------|
| DDL - 데이터 정의 언어 (Data Definition Language) | 데이터를 정의하기 위한 언어이다. 관계형 데이터베이스 구조(테이블, 스키마)를 정의하기 위한 명령어이다. | CREATE DROP ALTER |
| DML - 데이터 조작 언어 (Data Manipulation Language) | 데이터를 저장, 수정, 삭제, 조회 등을 하기 위한 언어이다. | INSERT UPDATE DELETE SELECT |
| DCL - 데이터 제어 언어 (Data Control Language) | 데이터베이스 사용자의 권한 제어를 위해 사용되는 언어이다. | GRANT REVOKE COMMIT ROLLBACK |

2. Hello, DB!


```
$ sqlite3
```

```
HPHK:~/workspace $ sqlite3  
SQLite version 3.8.2 2013-12-06 14:53:30  
Enter ".help" for instructions  
Enter SQL statements terminated with a ";"  
sqlite> █
```

<https://zzu.li/helloadb>

```
sqlite> .mode csv
```

CSV 파일을 가지고 와서 database로 만들어보자!

hellodb

| id | first_name | last_name | age | country | phone |
|----|------------|-----------|-----|---------|---------------|
| 1 | 길동 | 홍 | 600 | 충청도 | 010-2424-1232 |

1. Hello, World! Hello, SQL!

```
sqlite> SELECT * FROM examples;
```

1. Hello, World! Hello, SQL!

```
sqlite> SELECT * FROM examples;
```

```
sqlite> SELECT * FROM examples;  
1, "길동", "홍", 600, "충청도", 010-2424-1232
```

1. Hello, World! Hello, SQL!

```
sqlite> SELECT * FROM examples;
```

키워드(SELECT문)

키워드

SELECT * FROM table;

1. Hello, World! Hello, SQL!

```
sqlite> SELECT * FROM examples;
```

SELECT * **FROM** table;

> SELECT문은 데이터베이스에서 특정한 테이블을 반환한다.

조금 더 예쁘게 보자!

```
sqlite> .headers on
```

```
sqlite> SELECT * FROM examples;
```

| id | first_name | last_name | age | country | phone |
|----|------------|-----------|-----|---------|---------------|
| 1 | 길동 | 홍 | 600 | 충청도 | 010-2424-1232 |

3. DB, Table 생성

1. database 생성

\$ sqlite3 database

해당하는 데이터베이스 파일을 만들고 sqlite에서 확인해볼 수 있다.

1. database 생성

```
$ sqlite3 tutorial.sqlite3  
sqlite> .databases
```

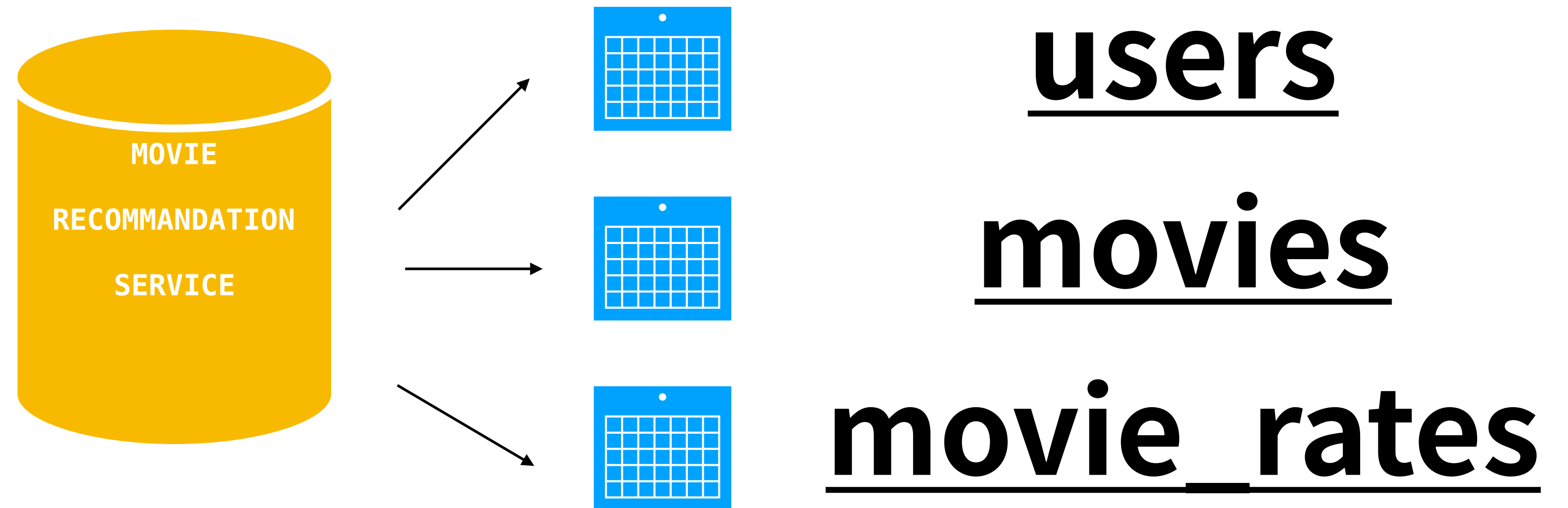
2. Table 생성

```
CREATE TABLE table (  
  column1 datatype PRIMARY KEY,  
  column2 datatype,  
  ....  
);
```

2. Table 생성

```
sqlite> CREATE TABLE classmates (  
        id INT PRIMARY KEY,  
        name TEXT  
        );
```

2. Table과 Database의 관계



2-1. Datatype

SQLite은 동적 데이터 타입으로, 기본적으로 Affinity 에 맞게 들어간다.
BOOLEAN은 정수 0, 1 으로 저장된다.

| | |
|----------|--|
| Affinity | |
| INTEGER | TINYINT(1byte), SMALLINT(2bytes), MEDIUMINT(3bytes), INT(4bytes), BIGINT(8bytes), UNSIGNED BIG INT |
| TEXT | CHARACTER(20), VARCHAR(255), TEXT |
| REAL | REAL, DOUBLE, FLOAT |
| NUMERIC | NUMERIC, DECIMAL, DATE, DATETIME |
| BLOB | <i>no datatype specified</i> |

2-2. Table 및 schema 조회

테이블 목록 조회

.tables

특정 테이블 스키마 조회

.schema table

2-2. Table 및 schema 조회

```
sqlite> .tables
```

```
sqlite> .schema classmates
```

2-3. Table 삭제 (DROP)

특정 table 삭제

DROP TABLE table;

2-3. Table 삭제 (DROP)

```
sqlite> DROP TABLE classmates;
```

```
sqlite> .tables
```

Q. 다음과 같은 스키마를 가지고 있는 classmate 테이블을 만들어보세요.

| column | datatype |
|---------|----------|
| id | INT |
| name | TEXT |
| age | INT |
| address | TEXT |

Q. 다음과 같은 스키마를 가지고 있는 classmate 테이블을 만들어보세요.

```
sqlite> CREATE TABLE classmates (  
        id INT PRIMARY KEY,  
        name TEXT,  
        age INT,  
        address TEXT );
```


4. 데이터 추가, 읽기, 수정, 삭제

1. data 추가 (INSERT)

특정 table에 새로운 행을 추가하여 데이터를 추가할 수 있습니다.

INSERT INTO table (column1, column2,..)

VALUES (value1, value2, ...);

1. data 추가 (INSERT)

Q. classmates 테이블에 이름이 홍길동이고 나이가 23인 데이터를 넣어봅시다!
그리고 SELECT문을 통해 확인해보세요 :)

2. Data 추가

Q. classmates 테이블에 이름이 홍길동이고 나이가 23인 데이터를 넣어봅시다!
그리고 SELECT문을 통해 확인해보세요 :)

```
sqlite> INSERT INTO classmates (name, age)  
VALUES ( '홍길동' , 23 );
```

1. data 추가 (INSERT)

Q. classmates 테이블에
id가 2이고,
이름이 홍길동이고,
나이가 30이고,
주소가 서울인 데이터를 넣어봅시다!

그리고 SELECT문을 통해 확인해보세요 :)

1. Data 추가

```
sqlite> INSERT INTO classmates  
VALUES (2, '홍길동', 30, '서울');
```

1. data 추가 (INSERT)

모든 열에 데이터를 넣을 때에는 column을 명시할 필요가 없습니다 :)

INSERT INTO table **VALUES** (value1, value2, ...)

Q. 이렇게 데이터를 저장하는 것이 맞을까..?

```
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|-------|-------|-------|---------|
| ----- | ----- | ----- | ----- |
| | 홍길동 | 23 | |
| 2 | 홍길동 | 30 | 서울 |

NO!

주소가 꼭 필요한 정보라면 공백으로 비워두면 안된다.

```
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|-------|-------|-------|---------|
| ----- | ----- | ----- | ----- |
| | 홍길동 | 23 | |
| 2 | 홍길동 | 30 | 서울 |

NO!

id는 Primary Key이므로 반드시 필요하며,
값이 저장되면 자동으로 증가하도록 한다.(unique)

```
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|-------|-------|-------|---------|
| ----- | ----- | ----- | ----- |
| | 홍길동 | 23 | |
| 2 | 홍길동 | 30 | 서울 |

* TABLE 설정 변경

Q. 다음과 같이 만들어봅시다.

```
sqlite> DROP TABLE classmates;  
  
sqlite> CREATE TABLE classmates (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name TEXT NOT NULL,
```

*주의! AUTOINCREMENT는
INTEGER 에서만 사용가능합니다.

* TABLE 설정 변경

```
sqlite> INSERT INTO classmates (name, age)
...> VALUES ('홍길동', 23);
Error: NOT NULL constraint failed: classmates.address
```

```
sqlite> INSERT INTO classmates (name, age, address) VALUES ('홍길동', 30, '서울');
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|----|------|-----|---------|
| 2 | 홍길동 | 30 | 서울 |
| 3 | 홍길동 | 30 | 서울 |

```
sqlite> INSERT INTO classmates VALUES (3, '홍길동', 50, '서울');
Error: UNIQUE constraint failed: classmates.id
```

2. data 가져오기 (SELECT)

REMIND!

SELECT * FROM table;

2. data 가져오기 (SELECT)

특정한 table에서 특정 Column만 가져오기

SELECT column1, column2 **FROM** table;

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값만 가져온다면?

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값만 가져온다면?

```
sqlite> SELECT id, name FROM classmates;
```


2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값을 몇개만 가져온다면?

SELECT column1, column2 FROM table

LIMIT num;

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값을 하나만 가져온다면?

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값을 하나만 가져온다면?

```
sqlite> SELECT id, name FROM classmates LIMIT 1;
```

2. data 가져오기 (SELECT)

Q. classmates에서 특정 column 값을 특정 위치에서부터 몇개만 가져온다면?

SELECT column1, column2 FROM table

LIMIT num **OFFSET** num;

LIMIT 와 OFFSET 은

세트입니다.

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값을 세번째에 있는 값 하나만 가져온다면?

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값을 세번째에 있는 값 하나만 가져온다면?

```
sqlite> SELECT id, name FROM classmates LIMIT 1 OFFSET 2;
```

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값 중에 특정한 값만 가져온다면?

SELECT column1, column2 FROM table

WHERE column=value;

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값 중에 주소가 서울인 사람만 가져온다면?

2. data 가져오기 (SELECT)

Q. classmates에서 id, name column 값 중에 주소가 서울인 사람만 가져온다면?

```
sqlite> SELECT id, name FROM classmates WHERE address="서울";
```

2. data 가져오기 (SELECT)

Q. classmates에서 특정 column 값을 중복없이 가져온다면?

SELECT DISTINCT column **FROM** table;

2. data 가져오기 (SELECT)

Q. classmates에서 age 값 전체를 중복없이 가져온다면?

```
sqlite> SELECT DISTINCT age FROM classmate;
```

3. data 삭제 (DELETE)

특정 table에 특정한 레코드를 삭제할 수 있습니다.

DELETE FROM table

WHERE condition;

Q. 무엇을 기준으로 삭제할까?

```
sqlite> INSERT INTO classmates (name, age, address) VALUES ('홍길동', 30, '서울');  
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|----|------|-----|---------|
| 2 | 홍길동 | 30 | 서울 |
| 3 | 홍길동 | 30 | 서울 |

중복이 불가능한(UNIQUE한)

값인 id를 기준으로 하자!

DELETE FROM table

WHERE id=?;

3. Data 삭제(DELETE)

Q. classmates 테이블에 id가 3인 레코드를 삭제 해봅시다.

3. Data 삭제(DELETE)

Q. classmates 테이블에 id가 3인 레코드를 삭제 해봅시다.

```
sqlite> DELETE FROM classmates WHERE id=3;
```


AUTOINCREMENT!

```
sqlite> DELETE FROM classmates
...> WHERE id=3;
sqlite>
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|----|------|-----|---------|
| 2 | 홍길동 | 30 | 서울 |

```
sqlite> INSERT INTO classmates (name, age, address) VALUES ('홍길동', 45, '서울');
sqlite> SELECT * FROM classmates;
```

| id | name | age | address |
|----|------|-----|---------|
| 2 | 홍길동 | 30 | 서울 |
| 4 | 홍길동 | 45 | 서울 |

4. data 수정 (UPDATE)

특정 table에 특정한 레코드를 수정할 수 있습니다.

UPDATE table

SET column1=value1, column2=value2, ...

WHERE condition;

4. data 수정 (UPDATE)

Q. classmates 테이블에 id가 4인 레코드를 수정해봅시다.

이름을 홍길동으로,
주소를 제주도로 바꿔보세요!

4. data 수정 (UPDATE)

Q. classmates 테이블에 id가 4인 레코드를 수정해봅시다.

이름을 홍길동으로,
주소를 제주로 바꿔보세요!

```
sqlite> UPDATE classmates  
...> SET name="홍길동", address="제주"  
...> WHERE id=4;
```

4. 데이터 추가, 읽기, 수정, 삭제 정리

| | 구문 | 예시 |
|---|--------|--|
| C | INSERT | INSERT INTO classmates (name, age, address) VALUES ('홍길동', '30', '서울'); |
| R | SELECT | SELECT * FROM classmates WHERE id=1; |
| U | UPDATE | UPDATE classmates SET name="철수" WHERE id=1; |
| D | DELETE | DELETE FROM classmates WHERE id=1; |

5. WHERE, expression

준비

<https://zzu.li/hellodb>

CSV 파일을 가지고 와서 database로 만들어보자!

| id | first_name | last_name | age | country | phone | balance |
|----|------------|-----------|-----|---------|---------------|---------|
| 1 | 정호 | 유 | 40 | 전라북도 | 016-7280-2855 | 370 |
| 2 | 경희 | 이 | 36 | 경상남도 | 011-9854-5133 | 5900 |
| 3 | 정자 | 구 | 37 | 전라남도 | 011-4177-8170 | 3100 |
| 4 | 미경 | 장 | 40 | 충청남도 | 011-9079-4419 | 250000 |
| 5 | 영환 | 차 | 30 | 충청북도 | 011-2921-4284 | 220 |
| 6 | 서준 | 이 | 26 | 충청북도 | 02-8601-7361 | 530 |
| 7 | 주원 | 민 | 18 | 경기도 | 011-2525-1976 | 390 |
| 8 | 예진 | 김 | 33 | 충청북도 | 010-5123-9107 | 3700 |
| 9 | 서현 | 김 | 23 | 제주특별자치도 | 016-6839-1106 | 43000 |
| 10 | 서윤 | 오 | 22 | 충청남도 | 011-9693-6452 | 49000 |
| 11 | 서영 | 김 | 15 | 제주특별자치도 | 016-3046-9822 | 640000 |
| 12 | 미정 | 류 | 22 | 충청남도 | 016-4336-8736 | 52000 |
| 13 | 하은 | 남 | 32 | 전라북도 | 016-9544-1490 | 35000 |
| 14 | 영일 | 김 | 35 | 전라남도 | 011-4448-6198 | 720 |
| 15 | 지원 | 박 | 24 | 경상북도 | 02-3783-1183 | 35000 |
| 16 | 옥자 | 김 | 19 | 경상남도 | 011-1038-5964 | 720 |
| 17 | 병철 | 고 | 34 | 충청남도 | 016-2455-8207 | 440 |
| 18 | 광수 | 김 | 17 | 충청북도 | 016-4058-7601 | 94000 |
| 19 | 성민 | 김 | 26 | 충청남도 | 011-6897-4723 | 6100 |
| 20 | 정수 | 김 | 17 | 경기도 | 016-1159-3227 | 590 |
| 21 | 동현 | 신 | 36 | 경상북도 | 010-1172-2541 | 4700 |
| 22 | 은정 | 황 | 16 | 강원도 | 016-5956-2725 | 7000 |
| 23 | 서준 | 김 | 26 | 강원도 | 02-4610-2333 | 6900 |
| 24 | 숙자 | 권 | 33 | 경상남도 | 016-4610-3200 | 230 |
| 25 | 유진 | 이 | 24 | 경기도 | 010-2349-9997 | 270000 |
| 26 | 영식 | 이 | 39 | 경상북도 | 016-2645-6128 | 400000 |
| 27 | 진호 | 백 | 17 | 경상남도 | 011-3885-5678 | 18000 |
| 28 | 성현 | 박 | 40 | 경상남도 | 011-2884-6546 | 580000 |
| 29 | 준서 | 서 | 36 | 충청남도 | 011-8419-5766 | 44000 |
| 30 | 영수 | 박 | 37 | 제주특별자치도 | 010-1106-3465 | 35000 |

```
sqlite> .mode csv
```

1. WHERE 심화

특정한 table에서 특정 조건의 Column만 가져오기

SELECT * FROM table

WHERE condition;

1. WHERE 심화

Q. users에서 age가 30 이상인 사람만 가져온다면?

1. WHERE 심화

Q. users에서 age가 30 이상인 사람만 가져온다면?

```
sqlite> SELECT * FROM users WHERE age >= 30;
```

1. WHERE 심화

Q. users에서 age가 30 이상인 사람의 이름만 가져온다면?

1. WHERE 심화

Q. users에서 age가 30 이상인 사람의 이름만 가져온다면?

```
sqlite> SELECT first_name FROM users WHERE age >= 30;
```

1. WHERE 심화

Q. users에서 age가 30 이상이고 성이 김인 사람의 성과 나이만 가져온다면?

1. WHERE 심화

Q. users에서 age가 30 이상이고 성이 김인 사람의 성과 나이만 가져온다면?

```
sqlite> SELECT age, last_name FROM users  
        WHERE age >= 30 and last_name="김";
```

2. Expression

다음의 표현식은 레코드의 개수를 반환한다.

SELECT **COUNT**(column) FROM table;

2. Expression

Q. users의 총 갯수는?

2. Expression

Q. users의 총 갯수는?

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

2. Expression

다음의 표현식은 기본적으로 숫자(INTEGER)일때만 가능하다.

SELECT **AVG(column)** FROM table;

AVG(), SUM(), MIN(), MAX()

2. Expression

Q. 30살 이상인 사람들의 평균나이?

2. Expression

Q. 30살 이상인 사람들의 평균나이?

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

2. Expression

Q. users에서 계좌 잔액(balance)이 가장 높은 사람과 액수는?

2. Expression

Q. users에서 계좌 잔액(balance)이 가장 높은 사람과 액수는?

```
sqlite> SELECT first_name, MAX(balance) FROM users;
```

2. Expression

Q. users에서 30살 이상인 사람의 계좌 평균 잔액은?

2. Expression

Q. users에서 30살 이상인 사람의 계좌 평균 잔액은?

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


3. LIKE

정확한 값에 대한 비교가 아닌, 패턴을 확인하여 해당하는 값을 반환한다.

SELECT * FROM table

WHERE column LIKE ‘’;

3. LIKE

WHERE column LIKE ‘’;

| | | |
|---|--------------|---------------------------|
| | | |
| % | 2% | 2로 시작하는 값 |
| | %2 | 2로 끝나는 값 |
| | %2% | 2가 들어가는 값 |
| _ | _2% | 아무값이나 들어가고 두번째가 2로 시작하는 값 |
| | 1__ | 1로 시작하고 4자리인 값 |
| | 2_%_% / 2__% | 2로 시작하고 적어도 3자리인 값 |

2. Expression

Q. users에서 20대인 사람의 테이블은?

2. Expression

Q. users에서 20대인 사람의 테이블은?

```
sqlite> SELECT * FROM users WHERE age LIKE '2%';
```

2. Expression

Q. users에서 지역번호가 02 인 사람만?

2. Expression

Q. users에서 지역번호가 02 인 사람만?

```
sqlite> SELECT * FROM users WHERE phone LIKE '02-%';
```

2. Expression

Q. users에서 이름이 준으로 끝나는 사람만?

2. Expression

Q. users에서 이름이 준으로 끝나는 사람만?

```
sqlite> SELECT * FROM users WHERE first_name LIKE '%준';
```


2. Expression

Q. users에서 중간 번호가 5114 인 사람만?

2. Expression

Q. users에서 중간 번호가 5114 인 사람만?

```
sqlite> SELECT * FROM users WHERE phone LIKE '%5114%';
```

6. ORDER

1. 정렬(ORDER)

SELECT columns FROM table

ORDER BY column1, column2 **ASC|DESC**;

ASC : 오름차순 (default)

DESC : 내림차순

1. 정렬(ORDER)

Q. users에서 나이순으로 오름차순 정렬하여 상위 10개만 뽑아보면?

1. 정렬(ORDER)

Q. users에서 나이순으로 오름차순 정렬하여 상위 10개만 뽑아보면?

```
sqlite> SELECT * FROM users ORDER BY age ASC LIMIT 10;
```

1. 정렬(ORDER)

Q. users에서 나이순, 성 순으로 오름차순 정렬하여 상위 10개만 뽑아보면?

1. 정렬(ORDER)

Q. users에서 나이순, 성 순으로 오름차순 정렬하여 상위 10개만 뽑아보면?

```
sqlite> SELECT * FROM users ORDER BY age, last_name ASC LIMIT 10;
```


1. 정렬(ORDER)

Q. users에서 계좌잔액순으로 내림차순 정렬하여 해당하는 사람이름 10개만 뽑아보면?

1. 정렬(ORDER)

Q. users에서 계좌잔액순으로 내림차순 정렬하여 해당하는 사람이름 10개만 뽑아보면?

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