

# **Database**

# 데이터베이스(DB)

데이터베이스는 체계화된 데이터의 모임이다.

여러 사람이 공유하고 사용할 목적으로 통합 관리되는 정보의 집합이다.

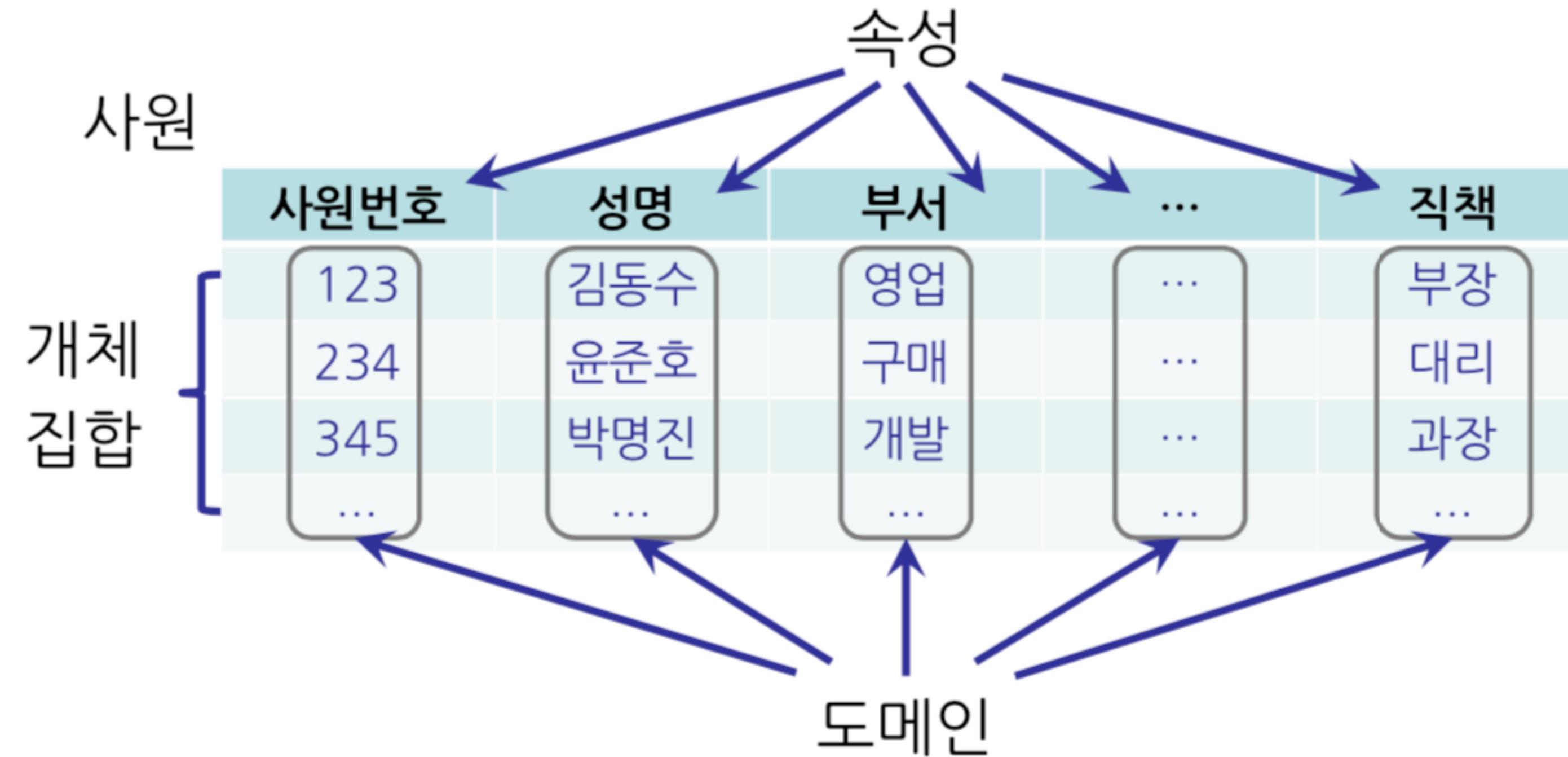
논리적으로 연관된 하나 이상의 자료의 모음으로 그 내용을 고도로 구조화함으로써 검색과 간신의 효율화를 꾀한 것이다.

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

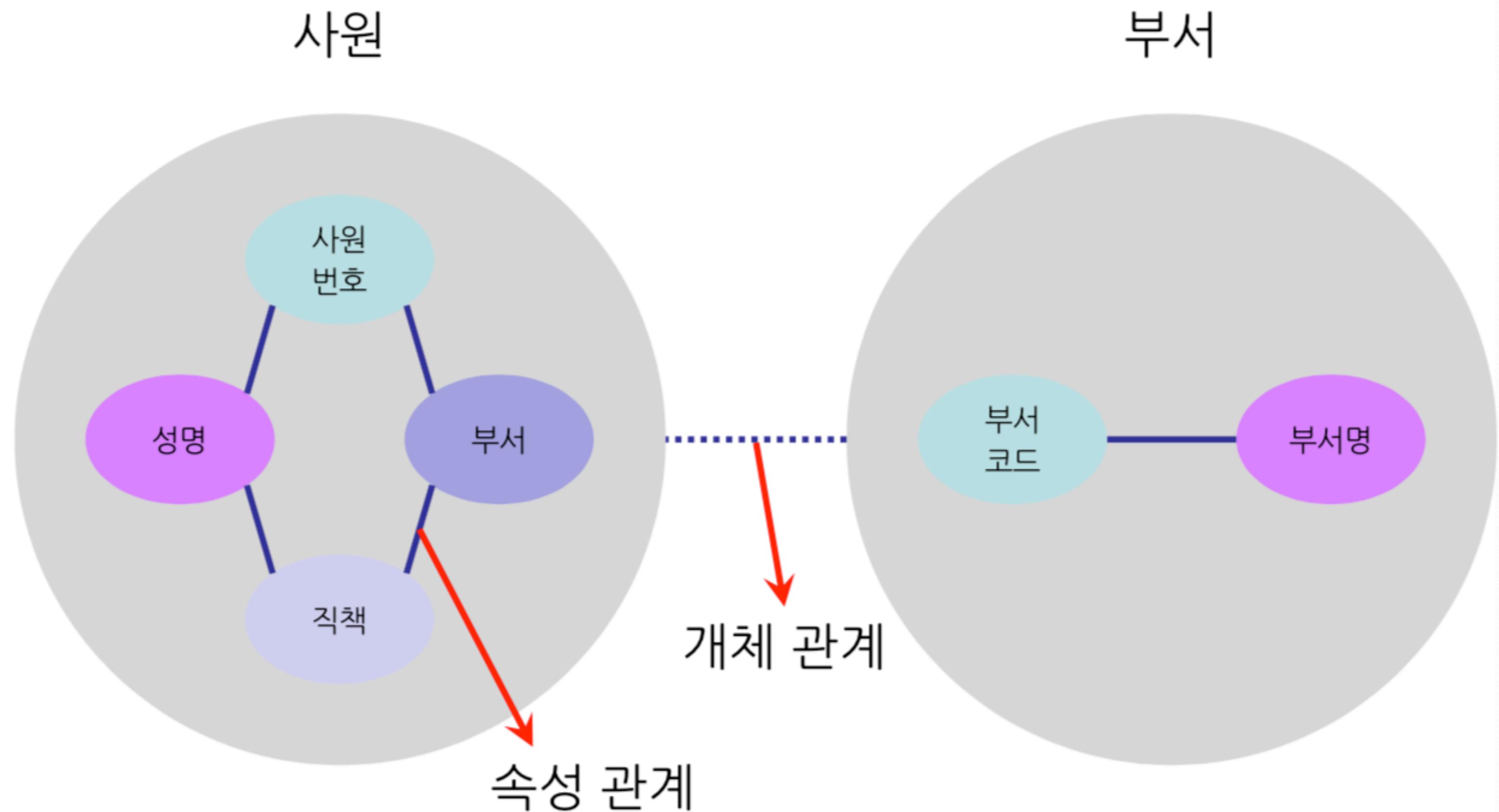
출처: 위키피디아

# 데이터베이스의 구성 요소

개체(entity)와 그들이 가지는 속성(attribute), 그리고 개체 사이의 관계(relation)

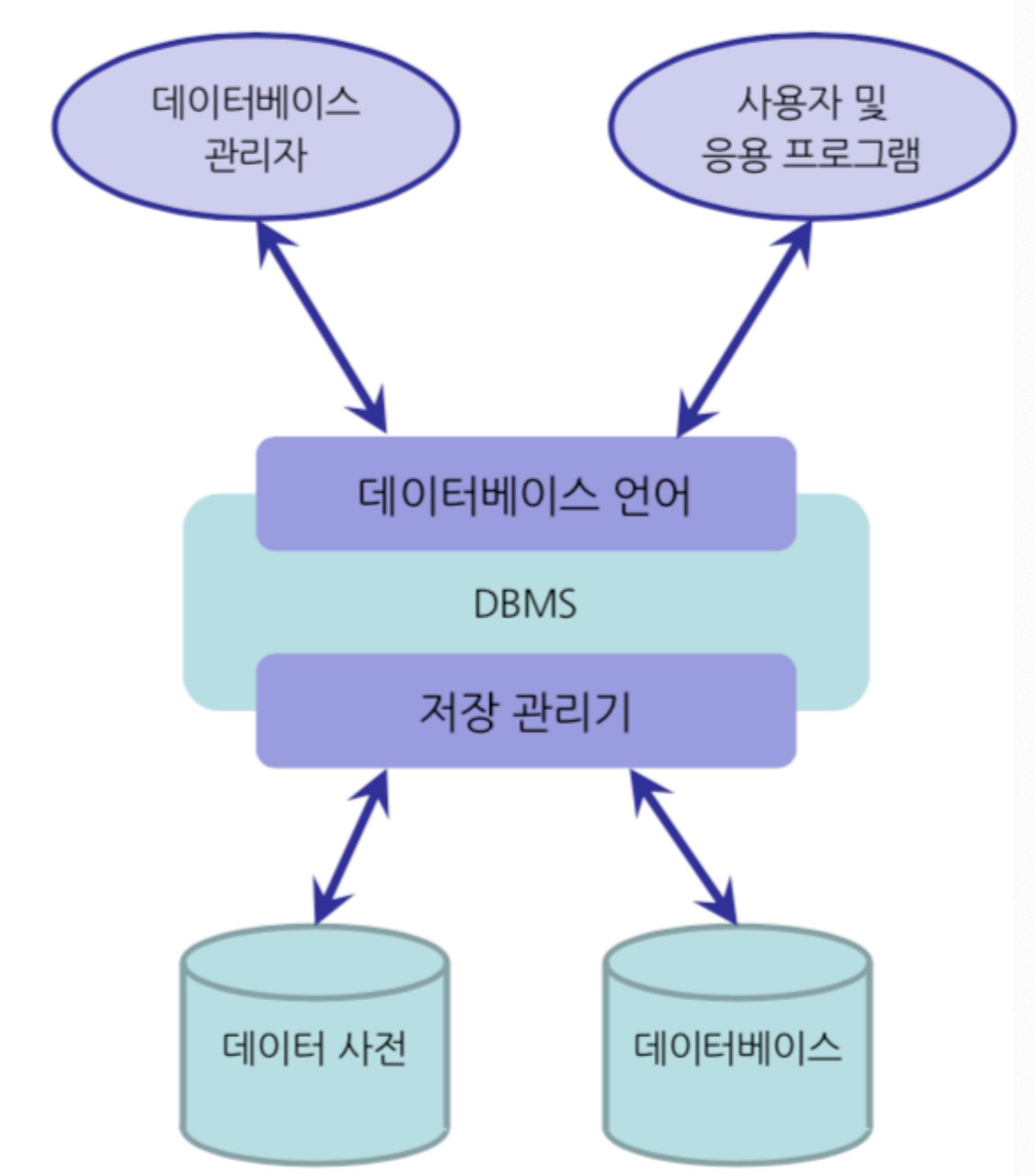


# 개체 관계와 속성 관계



# 데이터베이스로 얻는 장점들

- 데이터 중복 최소화
- 데이터 무결성
  - 정확한 정보를 보장
- 데이터 일관성
- 데이터 독립성
  - 물리적 독립성과 논리적 독립성
- 데이터 표준화
- 데이터 보안 유지



# RDBMS (관계형 데이터베이스 관리 시스템)

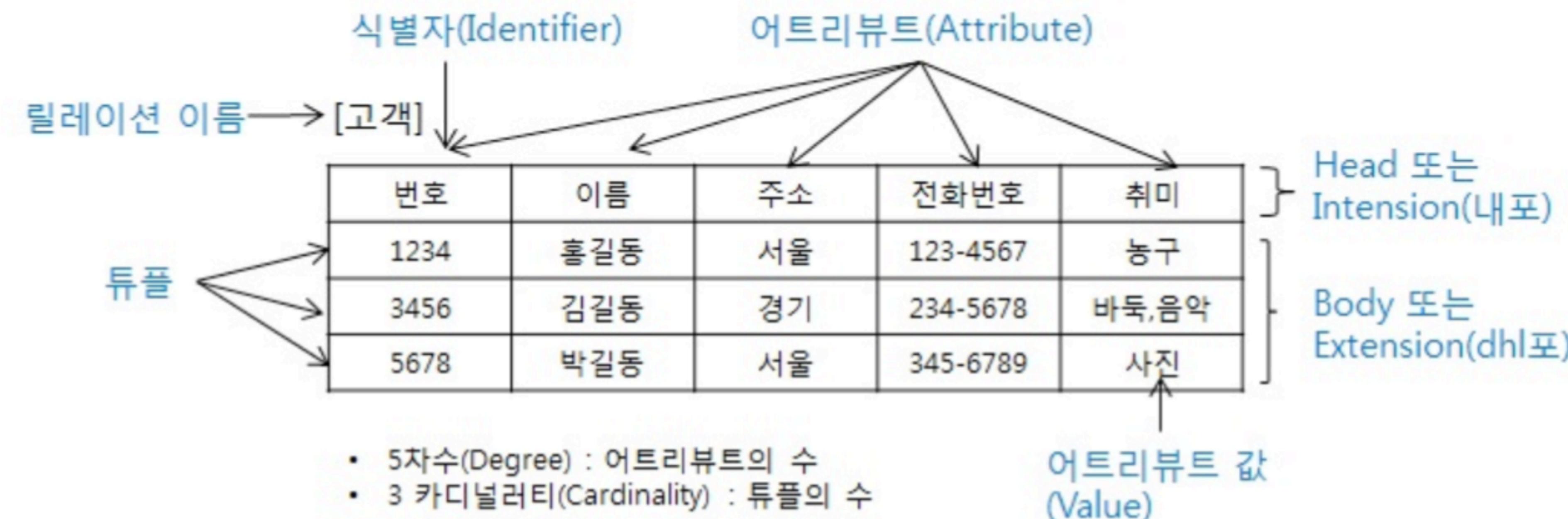
관계형 모델을 기반으로하는 데이터베이스 관리시스템이다.

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



# 관계형 데이터베이스 (Relational DB)

개체와 개체 사이의 관계를 표현하기 위하여 2차원의 표(table)를 사용



# RDBMS 특징

모든 데이터를 2차원으로 표현

테이블은 row(record, tuple)와 column(field, item)으로 구성

테이블은 상호 연관성을 지니고 하나의 DB에 여러 개 존재 가능

데이터베이스의 설계도를 ER(Entity Relationship) 모델이라고 하고  
ER모델에 따라 DB가 만들어짐



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		
Aug 2019	Jul 2019	Aug 2018			Aug 2019	Jul 2019	Aug 2018
1.	1.	1.	Oracle 	Relational, Multi-model 	1339.48	+18.22	+27.45
2.	2.	2.	MySQL 	Relational, Multi-model 	1253.68	+24.16	+46.87
3.	3.	3.	Microsoft SQL Server 	Relational, Multi-model 	1093.18	+2.35	+20.53
4.	4.	4.	PostgreSQL 	Relational, Multi-model 	481.33	-1.94	+63.83
5.	5.	5.	IBM Db2 	Relational, Multi-model 	172.95	-1.19	-8.89
6.	6.	6.	Microsoft Access	Relational	135.33	-1.98	+6.24
7.	7.	7.	SQLite 	Relational	122.72	-1.91	+8.99
8.	8.	↑ 9.	MariaDB 	Relational, Multi-model 	84.95	+0.52	+16.66
9.	9.	↑ 11.	Hive 	Relational	81.80	+0.93	+23.86
10.	10.	↓ 8.	Teradata 	Relational, Multi-model 	76.64	-1.18	-0.77

# 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 조건(자료의 구조, 표현 방법, 관계)에 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					

테이블(table) 열(컬럼/필드)과 행(레코드/값)의 모델을 사용해 조직된 데이터 요소들의 집합.  
SQL 데이터베이스에서는 테이블을 관계라고도 한다.

## 열(Column), 컬럼

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					

Diagram illustrating the concept of columns in a database. A vertical arrow points from the word "Column" to the second column of the table, which is highlighted in red. The table has six columns labeled A through E. Column A contains numerical values (1 to 5). Column B contains names ('hong', 'kim', 'kang', 'choi'). Column C contains ages (42, 16, 29, 8). Column D contains phone numbers. Column E contains email addresses. The first row serves as the header.

## 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					

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

The diagram illustrates a primary key constraint on a database table. A vertical line with a downward arrow points from the text "각 행(레코드)의 고유값으로 Primary Key로 불린다." to the "id" column of the table. The "id" column is highlighted with a red background. The table has six rows, indexed 1 to 6. The columns are labeled A, B, C, D, and E. Row 1 contains the header "id". Rows 2 through 5 contain data: (1, hong, 42, 010-1234-1234, hong@gmail.com), (2, kim, 16, 010-1234-5678, kim@naver.com), (3, kang, 29, 010-1111-2222, kang@hanmail.net), and (4, choi, 8, 010-3333-4444, choi@hotmail.com). Row 6 is empty. The bottom navigation bar shows "user" and "product" buttons.

A	B	C	D	E
1	id	name	age	phone
2	1	hong	42	010-1234-1234
3	2	kim	16	010-1234-5678
4	3	kang	29	010-1111-2222
5	4	choi	8	010-3333-4444
6				

PK(기본키)

열(Column)

행(row), 레코드

	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					

테이블(table)

# 1. SQL 개념

# SQL

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

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.

분류	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

# SQL 특징

- ; 까지 하나의 명령으로 간주

- .은 sqlite3 프로그램 기능 실행

- .schema 테이블이름 -> ;를 붙이지 않음

- 테이블은 여러 개 존재 가능

- 소문자로 표현해도 됨. 단, 대문자를 강력하게 권장함

# SQL Keywords

테이블에 데이터 삽입 (새로운 행 추가) -> INSERT

데이터 삭제 (행 제거) -> DELETE

데이터 갱신 -> UPDATE

데이터 검색 -> SELECT

# **2. Hello, DB!**

**[https://bit.do/hello\\_db](https://bit.do/hello_db)**

**sqlite.zip** 다운로드

**C:\Users\student (~)에 압축풀기**

# 다음과 같은 파일/폴더 구조로 만들기

C:

  └ Users

    └ student

      └ sqlite

      └ sqlite3.def

      └ sqlite3(.exe)

windows key => 계정의 환경 변수 편집

student에 대한 사용자 변수 => Path 더블클릭

**새로 만들기 클릭**

**C:\Users\student\sqlite**

**입력 후 모두 확인!**

**vscode 종료 후 재실행!**

**! vscode 터미널에서**



**\$ sqlite3**

!종료할 때

**ctrl + Z => Enter 또는 .exit 입력**

```
sqlite> ^Z (enter)
```

\$

# CSV 파일을 가지고 와서 database로 만들어보자!

hellobd

<b>id</b>	<b>first_name</b>	<b>last_name</b>	<b>age</b>	<b>country</b>	<b>phone</b>
1	길동	홍	600	충청도	010-2424-1232

# **3. Table 과 DB**

**db 실습을 진행 할 프로젝트 폴더로 이동**

**00\_db\_prac\_1**

# Database 생성

**\$ sqlite3 database**

해당하는 데이터베이스 파일이 있으면 해당 DB를 콘솔로 연다.

만약 해당하는 파일이 없으면 새로 생성하고, 해당 DB를 콘솔로 연다.

# Database 생성

```
$ sqlite3 tutorial.sqlite3
```

```
sqlite> .databases
```

**[https://bit.do/hello\\_db](https://bit.do/hello_db)**

**hellobd.csv db 프로젝트 폴더 안에 다운로드**

```
sqlite> .mode csv
```

```
sqlite> .import hellobd.csv examples
```

**CSV 파일을 가지고 와서  
examples라는 table로 만들어보자!**

hellobd

<b><u>id</u></b>	<b><u>first_name</u></b>	<b><u>last_name</u></b>	<b><u>age</u></b>	<b><u>country</u></b>	<b><u>phone</u></b>
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
```

키워드(SELECT문)

키워드

**SELECT \* FROM table;**

# **SELECT \* FROM table;**

> SELECT문은 데이터베이스에서 특정한 테이블을 반환한다.

## 조금 더 예쁘게 보자!

```
sqlite> .headers on  
sqlite> .mode column
```

```
sqlite> .headers on  
sqlite> .mode column  
sqlite> SELECT * FROM examples;  
id      first_name  last_name    age      country    phone  
-----  -----  -----  -----  
1       길동        홍          600      충청도    010-2424-1232
```

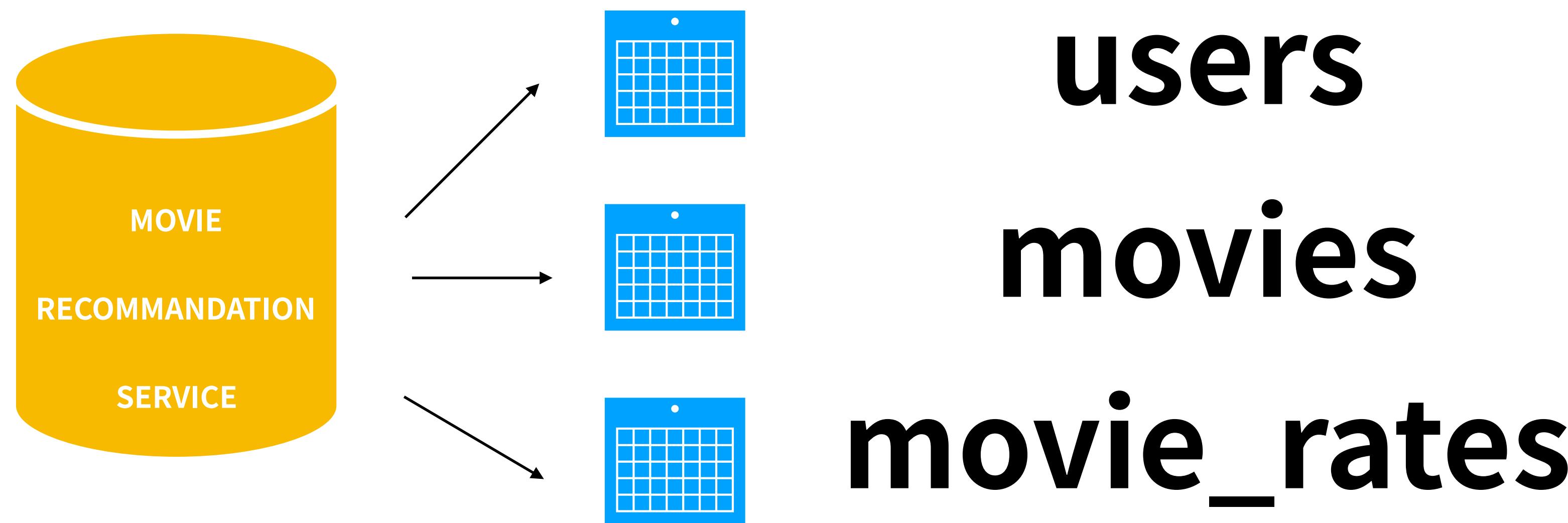
# Table 생성

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

# Table 생성

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

# Table 과 DB 의 관계



# Datatype

SQLite은 동적 데이터 타입으로, 기본적으로 유연하게 데이터가 들어간다.

BOOLEAN은 따로 타입이 존재하지 않기 때문에 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, BOOLEAN, DATE, DATETIME
BLOB	no datatype specified

# Table 및 Schema 조회

테이블 목록 조회

**.tables**

특정 테이블 스키마 조회

**.schema table**

# Table 및 Schema 조회

```
sqlite> .tables
```

```
sqlite> .schema classmates
```

# Table 삭제(DROP)

특정 table 삭제

**DROP TABLE table;**

# Table 삭제(DROP)

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

```
sqlite> .tables
```

**Q. 다음과 같은 스키마를 가지고 있는 `classmates` 테이블을 만들어보세요.**

column	datatype
name	TEXT
age	INT
address	TEXT

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

```
sqlite> .tables
```

```
sqlite> .schema classmates
```

# **4. 데이터 추가, 읽기, 수정, 삭제**

# Data 추가(INSERT)

특정 table에 새로운 행을 추가하여 데이터를 추가할 수 있습니다.

**INSERT INTO** table (column1, column2, ...)

**VALUES** (value1, value2, ...);

# Data 추가(INSERT)

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

# Data 추가(INSERT)

우선 테이블의 schema부터 파악

```
sqlite>.schema classmates
```

# Data 추가(INSERT)

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

# Data 추가(INSERT)

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

# Data 추가(INSERT)

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

그리고 SELECT문을 통해 확인해보세요 :)

# Data 추가(INSERT)

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

# Data 추가(INSERT)

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

모든 열에 데이터를 넣을 때에는 column을 명시할 필요가 없습니다.

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

Q. 여기서 잠깐! id 는 어디에 있을까?

```
sqlite> SELECT * FROM classmates;
name          age          address
-----        -----
홍길동        23
홍길동        23          서울
```

A. SQLite 는 따로 PRIMARY KEY 속성의 컬럼을 작성하지 않으면  
값이 자동으로 증가하는 PK 옵션을 가진 **rowid** 컬럼을 정의한다.

sqlite> SELECT rowid, * FROM classmates;			
rowid	name	age	address
1	홍길동	23	
2	홍길동	23	서울

- \* rowid 는 64bit 정수 타입의 유일한 식별 값이다.
- \* INTEGER PRIMARY KEY 타입으로 컬럼을 만들면 이는 rowid 를 대체한다.

Q. 여기서 또 잠깐! address 값이 없는 상태로 작성되는게 맞을까?

sqlite> SELECT * FROM classmates;		
name	age	address
-----	-----	-----
홍길동	23	
홍길동	23	서울

**NO!**

주소가 꼭 필요한 정보라면 공백으로 비워두면 안된다.

**(NOT NULL)**

이는 데이터 베이스 무결성의 원칙에 위배

# Table 다시 만들기

Q. 다음과 같이 만들어봅시다.

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

```
sqlite> .tables
```

```
sqlite> CREATE TABLE classmates (
```

```
    id INTEGER PRIMARY KEY,
```

```
    name TEXT NOT NULL,
```

```
    age INT NOT NULL,
```

```
    address TEXT NOT NULL
```

```
);
```

\*주의! PRIMARY KEY는  
INTEGER에서만 사용 가능합니다.

# Table 다시 만들기

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

```
Error: NOT NULL constraint failed: classmates.address
```

```
sqlite> INSERT INTO classmates VALUES ('홍길동', 30, '서울');
Error: table classmates has 4 columns but 3 values were supplied
```

```
sqlite> INSERT INTO classmates VALUES (1, '홍길동', 30, '서울');
sqlite> SELECT * FROM classmates;
id          name        age        address
-----      -----      -----      -----
1           홍길동     30         서울
```

`rowid` 는 원래 자동으로 작성 되었는데,  
직접 `id` 컬럼을 작성한 후에는 `id` 컬럼을 입력하지 않으면  
`rowid` 컬럼이 자동으로 입력되지 않는다.

```
sqlite> INSERT INTO classmates VALUES ('홍길동', 30, '서울');  
Error: table classmates has 4 columns but 3 values were supplied
```

그래서 PK 컬럼은 직접 작성하기보다  
SQLite 가 만들어주는 rowid 를 사용하는 것이 좋다.

# Table 다시x2 만들기

```
sqlite> DROP TABLE classmates;  
  
sqlite> .tables  
  
sqlite> CREATE TABLE classmates (  
    name TEXT NOT NULL,  
    age  INT NOT NULL,  
    address TEXT NOT NULL );  
  
sqlite> INSERT INTO classmates VALUES ('홍길동', 30, '서울'), ('김철수', 23,  
    '대전'), ('박나래', 23, '광주'), ('이요셉', 33, '구미');  
  
sqlite> SELECT rowid, * FROM classmates;
```

# Data 조회(SELECT)

REMIND!

**SELECT \* FROM table;**

# Data 조회(SELECT)

특정한 table에서 특정 Column만 가져오기

**SELECT column1, column2 FROM table;**

# Data 조회(SELECT)

Q. **classmates**에서 **id, name column** 값만 가져온다면?

# Data 조회(SELECT)

Q. **classmates**에서 **id, name column** 값만 가져온다면?

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

# Data 조회(SELECT)

특정한 table에서 원하는 개수만큼 Column 가져오기

```
SELECT column1, column2 FROM table  
      LIMIT num;
```

# Data 조회(SELECT)

Q. **classmates**에서 **id, name** column 값을 하나만 가져온다면 ?

# Data 조회(SELECT)

Q. **classmates**에서 **id, name column** 값을 하나만 가져온다면 ?

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

# Data 조회(SELECT)

Q. table에서 특정 column 값을 특정 위치에서부터 몇 개만 가져온다면 ?

```
SELECT column1, column2 FROM table  
LIMIT num OFFSET num;
```

LIMIT 와 OFFSET 은 한 세트!

# Data 조회(SELECT)

Q. **classmates**에서 **id, name** column 값을 세번째에 있는 값 하나만 가져온다면 ?

# Data 조회(SELECT)

Q. **classmates**에서 **id, name column** 값을 세번째에 있는 값 하나만 가져온다면 ?

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

# Data 조회(SELECT)

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

SELECT column1, column2 FROM table  
**WHERE column=value;**

# Data 조회(SELECT)

Q. **classmates**에서 **id, name column** 값 중에 주소가 서울인 사람만 가져온다면 ?

# Data 조회(SELECT)

Q. **classmates**에서 **id, name column** 값 중에 주소가 서울인 사람만 가져온다면 ?

```
sqlite> SELECT rowid, name FROM classmates WHERE address='서울';
```

# Data 조회(SELECT)

table에서 특정 column 값을 중복없이 가져오기

**SELECT DISTINCT column FROM table;**

# Data 조회(SELECT)

Q. **classmates**에서 age 값 전체를 중복없이 가져온다면?

# Data 조회(SELECT)

Q. 현재 들어가있는 값을 먼저 확인해보자

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

# Data 조회(SELECT)

Q. **classmates**에서 **age** 값 전체를 중복없이 가져온다면?

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

# Data 삭제(DELETE)

특정 table에 특정한 레코드를 삭제할 수 있습니다.

**DELETE FROM table**

**WHERE condition;**

# Data 삭제(DELETE)

Q. 무엇을 **기준**으로 삭제할까?

# Data 삭제(DELETE)

중복이 불가능한(UNIQUE)값인 **rowid**를 기준으로 하자!

**DELETE FROM table**

**WHERE rowid=?;**

# Data 삭제(DELETE)

Q. **classmates** 테이블에 id가 4인 레코드를 삭제 해봅시다.

# Data 삭제(DELETE)

Q. **classmates** 테이블에 id가 4인 레코드를 삭제 해봅시다.

```
sqlite> DELETE FROM classmates WHERE rowid=4;
```

# SQLite 는 기본적으로 일부 행을 삭제하고 새 행을 삽입하면 삭제 된 행의 값을 재사용하려고 시도한다.

```
sqlite> SELECT rowid, * FROM classmates;
rowid      name      age      address
-----  -----
1          홍길동    30      서울
2          김철수    23      대전
3          박나래    23      광주
sqlite> INSERT INTO classmates VALUES ('최철순', 45, '서울');
sqlite> SELECT rowid, * FROM classmates;
rowid      name      age      address
-----  -----
1          홍길동    30      서울
2          김철수    23      대전
3          박나래    23      광주
4          최철순    45      서울
```

Q. 이전에 삭제 된 행의 값을 재사용하지 않고  
사용하지 않은 값을 다음 행 값으로 사용하게 하려면 어떻게 해야 할까?

# **AUTOINCREMENT**

# Table 만들기

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

sqlite> .tables

sqlite> .schema tests

sqlite> INSERT INTO tests VALUES (1, '홍길동'), (2, '김철수');
```

```
sqlite> SELECT * FROM tests;
```

<b>id</b>	<b>name</b>
-----------	-------------

-----	-----
-------	-------

1	홍길동
---	-----

2	김철수
---	-----

```
sqlite> DELETE FROM tests WHERE id=2;
```

```
sqlite> SELECT * FROM tests;
```

<b>id</b>	<b>name</b>
-----------	-------------

-----	-----
-------	-------

1	홍길동
---	-----

```
sqlite> INSERT INTO tests (name) VALUES ('최철순');
```

```
sqlite> SELECT * FROM tests;
```

<b>id</b>	<b>name</b>
-----------	-------------

-----	-----
-------	-------

1	홍길동
---	-----

3	최철순
---	-----

하지만 SQLite 는  
특정한 요구사항(사용 되지 않은 값이나 이전에 삭제 된 행의 값을 재사용하지 못하게 하는)이 없다면  
**AUTOINCREMENT 속성을 사용하지 않아야 한다고 한다.**

\* 내부적으로 CPU, 메모리, 디스크 공간을 추가로 불필요하게 사용하므로  
엄격하게 필요하지 않을 경우 사용을 피해야 한다.

# Data 수정(UPDATE)

특정 table에 특정한 레코드를 수정할 수 있습니다.

**UPDATE** table

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

**WHERE** condition;

# Data 수정(UPDATE)

Q. **classmates** 테이블에 rowid가 4인 레코드를 수정해봅시다.  
이름을 홍길동으로, 주소를 제주도로 바꿔보세요!

# Data 수정(UPDATE)

그 전에 다시 `classmates` 테이블의 `schema`와 데이터 확인!

```
sqlite> .tables
```

```
sqlite> .schema classmates
```

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

# Data 수정(UPDATE)

Q. **classmates** 테이블에 rowid가 4인 레코드를 수정해봅시다.  
이름을 홍길동으로, 주소를 제주도로 바꿔보세요!

```
sqlite> UPDATE classmates  
        SET name='홍길동', address='제주'  
        WHERE rowid=4;
```

# Data 수정(UPDATE)

데이터가 수정 확인

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

# 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**

**[https://bit.do/hello\\_db](https://bit.do/hello_db)**

**users.csv db 프로젝트 폴더 안에 다운로드**

---

<b>id</b>	<b>first_name</b>	<b>last_name</b>	<b>age</b>	<b>country</b>	<b>phone</b>	<b>balance</b>
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

**users.csv 파일을 가지고  
database > users 테이블  
에 데이터를 추가해 보자.**

**sqlite> .mode csv**

**sqlite> .import users.csv users**

**sqlite> .tables**

**sqlite> SELECT \* FROM users;**

```
sqlite> .headers on
```

# WHERE 문 심화

특정한 table에서 특정 조건의 Column만 가져오기

**SELECT \* FROM table**

**WHERE condition;**

# WHERE 문 심화

Q. users에서 age가 30 이상인 사람만 가져온다면 ?

# WHERE 문 심화

Q. users에서 age가 30 이상인 사람만 가져온다면 ?

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

# WHERE 문 심화

Q. users에서 age가 30 이상인 사람의 이름만 가져온다면 ?

# WHERE 문 심화

Q. users에서 age가 30 이상인 사람의 이름만 가져온다면 ?

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

# WHERE 문 심화

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

# WHERE 문 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';
```

# Expressions

다음의 표현식은 레코드의 개수를 반환한다.

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

# Expressions

Q. users 테이블의 레코드 총 개수는 ?

# Expressions

Q. users 테이블의 레코드 총 개수는 ?

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

# Expressions

다음의 표현식은 기본적으로 숫자(INTEGER)일때만 가능하다.

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

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

# Expressions

Q. 30살 이상인 사람들의 평균나이는 ?

# Expressions

Q. 30살 이상인 사람들의 평균나이는 ?

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

# Expressions

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

# Expressions

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

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

# Expressions

Q. users에서 30살 이상인 사람의 계좌 평균 잔액은 ?

# Expressions

Q. users에서 30살 이상인 사람의 계좌 평균 잔액은 ?

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

# LIKE (wild cards)

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

```
SELECT * FROM table  
WHERE column LIKE “”;
```

# LIKE (wild cards)

## 와일드 카드 2가지 패턴

—

반드시 이 자리에  
한 개의 문자가 존재해야 한다.

%

이 자리에 문자열이  
있을 수도, 없을 수도 있다.

# LIKE (wild cards)

**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

# LIKE (wild cards)

시작하기 전! users 테이블의 schema 다시 확인!

```
sqlite> .schema users
```

# LIKE (wild cards)

Q. users에서 20대인 사람은?

# LIKE (wild cards)

Q. users에서 20대인 사람은?

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

# LIKE (wild cards)

Q. users에서 지역번호가 02 인 사람만 ?

# LIKE (wild cards)

Q. users에서 지역번호가 02 인 사람만 ?

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

# LIKE (wild cards)

Q. users에서 이름이 '준'으로 끝나는 사람만 ?

# LIKE (wild cards)

Q. users에서 이름이 '준'으로 끝나는 사람만 ?

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

# LIKE (wild cards)

Q. users에서 중간 번호가 5114 인 사람만 ?

# LIKE (wild cards)

Q. users에서 중간 번호가 5114 인 사람만 ?

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

# **6. ORDER**

# 정렬 (ORDER)

SELECT columns FROM table

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

ASC : 오름차순 (default)

DESC : 내림차순

# 정렬 (ORDER)

Q. users에서 나이순으로 오름차순 정렬하여 상위 10개만 뽑아보면 ?

# 정렬 (ORDER)

Q. users에서 나이순으로 오름차순 정렬하여 상위 10개만 뽑아보면 ?

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

# 정렬 (ORDER)

ASC(오름차순)은 default 설정이기 때문에 ASC를 적지 않아도 동작!

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

# 정렬 (ORDER)

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

# 정렬 (ORDER)

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

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

# 정렬 (ORDER)

ASC(오름차순)은 default 설정이기 때문에 ASC를 적지 않아도 동작!

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

# 정렬 (ORDER)

Q. users에서 계좌잔액순으로 내림차순 정렬하여  
해당하는 사람의 성과 이름을 10개만 뽑아보면 ?

# 정렬 (ORDER)

Q. users에서 계좌잔액순으로 내림차순 정렬하여  
해당하는 사람의 성과 이름을 10개만 뽑아보면 ?

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

# 정렬 (ORDER)

계좌 잔액까지 같이 보고 싶다면?!

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

# **7. ALTER**

## **7.1 테이블명 변경**

# 테이블 생성

Q. 먼저 새로운 테이블 `articles`를 생성해 보자.

`title: TEXT NOT NULL`

`content: TEXT NOT NULL`

# 테이블 생성

Q. 먼저 새로운 테이블 **articles**를 생성해 보자.

**title: TEXT NOT NULL**

**content: TEXT NOT NULL**

```
sqlite> CREATE TABLE articles (
    title TEXT NOT NULL,
    content TEXT NOT NULL
);
sqlite> .tables
```

# 데이터 삽입

Q. **articles** 테이블에 값을 넣어 보자

```
sqlite> INSERT INTO articles VALUES ('1번제목', '1번내용');
```

특정 테이블의 이름을 변경한다.

```
ALTER TABLE exist_table  
RENAME TO new_table;
```

# 테이블명 변경

```
sqlite> ALTER TABLE articles RENAME TO news;  
sqlite> .tables
```

## **7.2 새로운 컬럼 추가**

# 새로운 컬럼 추가

특정 테이블에 새로운 컬럼을 추가한다.

```
ALTER TABLE table  
ADD COLUMN col_name DATATYPE;
```

# 새로운 컬럼 추가

Q. 새로운 컬럼 `created_at`를 추가하고 확인해 보자.

`created_at: DATETIME`

```
sqlite> ALTER TABLE news  
        ADD COLUMN created_at DATETIME NOT NULL;
```

기존 데이터에 NOT NULL 조건으로 인해 NULL 값으로  
새로운 컬럼이 추가될 수 없으므로 아래와 같은 에러가 발생한다.  
NOT NULL 조건을 없애거나 기본값(DEFAULT)을 지정해야 한다.

**Cannot add NOT NULL column with  
default value NULL**

# 새로운 컬럼 추가 - 1

Q. NOT NULL 조건을 빼고 컬럼을 추가하면 정상적으로 추가된다.

```
sqlite> ALTER TABLE news
```

```
    ADD COLUMN created_at DATETIME;
```

```
sqlite> .schema news
```

```
sqlite> INSERT INTO news
```

```
    VALUES ('title', 'content', datetime('now', 'localtime'));
```

```
sqlite> SELECT rowid, * FROM news;
```

# 새로운 컬럼 추가 - 2

Q. 또는 DEFAULT 값을 넣어서 추가한다.

```
sqlite> ALTER TABLE news
```

```
    ADD COLUMN subtitle TEXT NOT NULL DEFAULT 1;
```

```
sqlite> .schema news
```

```
sqlite> SELECT * FROM news;
```