

10장. 데이터베이스 연동 (sqlite3)



데이터베이스

데이터베이스(Database)

- 데이터들이 모여있는 데이터의 집합으로 **서로 관련 있는 데이터들의 모임이다.**

(메모장에 두서없이 적어 놓은 단어들의 모임은 데이터베이스가 아님)

- 데이터베이스와 생활

예) 학교 홈페이지에서 수강신청, 성적 조회

전산화된 도서관에서 도서 검색, 비행기나 기차 예매 등

학생 테이블

| 학번 | 이름 | 생년월일 | 학과명 |
|----------|-----|-------------|--------|
| 20150001 | 오상식 | 1987. 6. 10 | 컴퓨터공학과 |
| 20171010 | 최정보 | 1995. 5. 5 | 전자공학과 |
| 20182121 | 김나래 | 1993. 12. 1 | 기계공학과 |

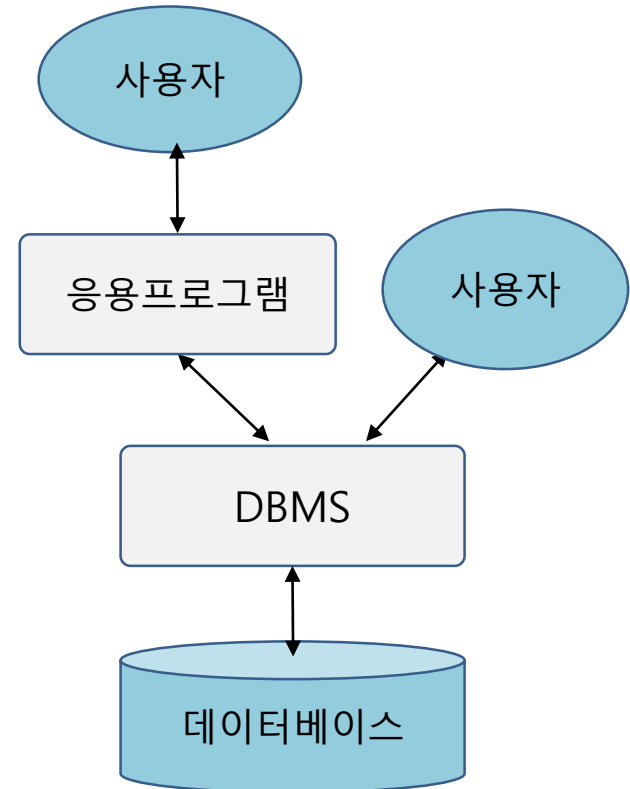
과목 테이블

| 과목번호 | 과목명 | 담당교수 |
|------|---------|------|
| 0303 | 웹 프로그래밍 | 송미영 |
| 0116 | 데이터베이스 | 오용철 |

데이터베이스 관리 시스템

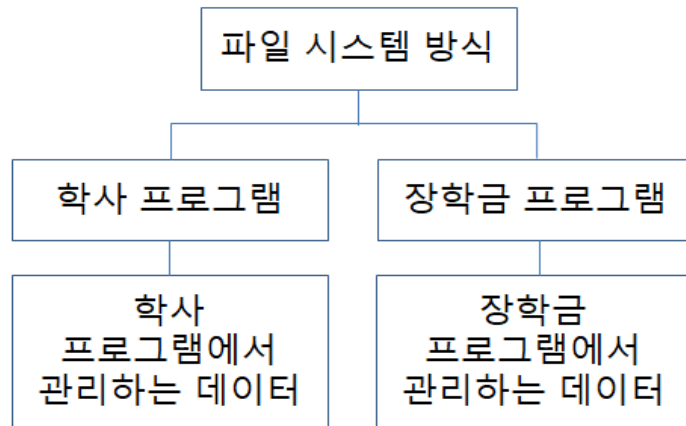
데이터베이스 관리 시스템(Database Management System)

- 많은 양의 데이터를 정교하게 구축하고 관리하는 소프트웨어이다.
- 데이터베이스의 정의, 데이터베이스 갱신, 질의 처리, 유지보수, 보안 등의 편리한 기능을 제공한다.
- 대표적으로 오라클 사의 Oracle 과 MySQL, 마이크로소프트사의 MSSQL등이 있다
한편 sqlite3는 DB 서버가 아닌 응용 프로그램에 넣어 사용하는 가벼운(경량) 데이터베이스이다.

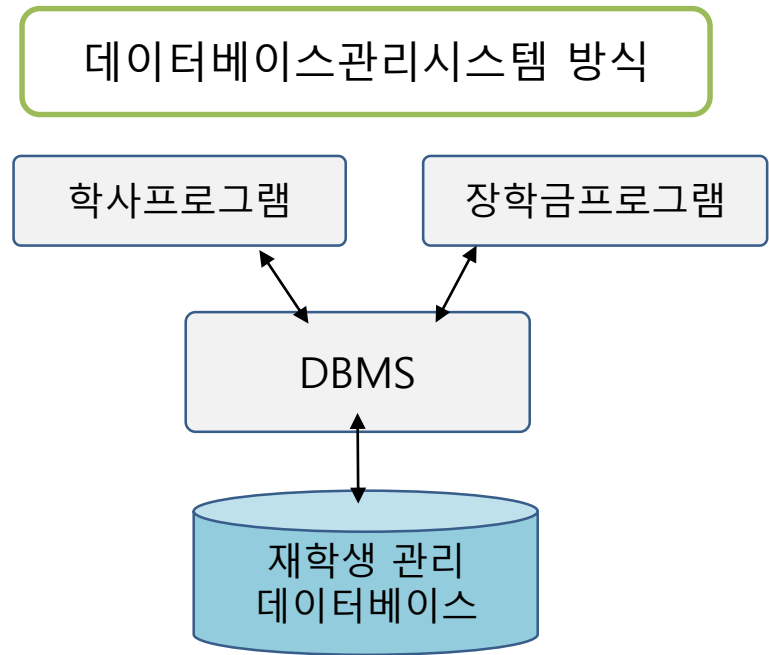


파일 시스템과 DBMS

파일시스템과 DBMS



파일 시스템은 서로 다른 여러 응용 프로그램이 제공하는 기능에 맞게 필요한 데이터를 각각 저장하고 관리한다. 따라서 각 파일에 저장한 데이터는 서로 연관이 없고 중복 또는 누락이 발생할 수 있다.



학생과 관련된 일련의 데이터를 한곳에 모아 관리하면 데이터의 오류, 누락, 중복 등의 문제를 해결할 수 있다.

파일 시스템과 DBMS

파일 시스템 방식의 문제점

이순신 학생이 졸업했는데 업데이트가 되지 않아 재학중으로 되어 있어 장학금 신청이 가능한 걸로 오류 발생

학사 프로그램

| 학번 | 이름 | 학과 | 상태 |
|-----------|-----|--------|-----|
| 2019-0001 | 홍길동 | 컴퓨터공학과 | 군휴학 |
| 2019-0002 | 이순신 | 경영학과 | 졸업 |
| 2019-0003 | 유관순 | 철학과 | 재학 |

장학금 신청 프로그램

| 장학금 | 이름 | 상태 | 가능여부 |
|-----|-----|-----|------|
| 국가 | 홍길동 | 군휴학 | 신청불가 |
| 성적 | 이순신 | 재학 | 신청가능 |
| 근로 | 유관순 | 재학 | 신청가능 |

데이터베이스 관리 시스템

데이터베이스 관리 시스템의 장점

- 데이터의 중복과 불일치 감소

데이터가 여러 곳에 분산되어 있으면 중복 저장될 수 있고, 같은 의미의 데이터가 다른 값을 갖게 되는 불일치가 생길 수 있다.

- 질의 처리에 효율적인 저장 구조

사용자는 질의(Query)를 통해서 데이터베이스에 접근하는데 시간이 소요되지만 DBMS는 시간을 줄이도록 저장 구조가 설계되어 있다.

- 백업(Backup)과 복구(Recovery)

데이터는 저장과 동시에 반드시 백업(따로 복사)되어야 한다. 복구는 트랜잭션(업무 단위)을 관리하여 데이터베이스가 피해를 보기 전 상태로 복구하는 것이다.

※ 단점 : 사용하는 자원이 많고 복잡하며 비싸다.

데이터 모델

데이터 모델

- 컴퓨터에 데이터를 저장하는 방식을 정의해 놓은 개념 모형이다.
- 계층형 데이터 모델, 네트워크형 데이터 모델, 관계형 데이터 모델, 객체 지향형

데이터 모델링(Data Modeling)

- 데이터 베이스의 설계시 클라이언트의 요구를 분석하여 논리모델을 구성하고 물리모델을 사용해 데이터베이스에 반영하는 작업
- 기본 요소

| 구분 | 개념 | 실제 예 |
|-------------------|---------------------------|----------------|
| 엔티티(Entity) | 물리적 개념에서는 테이블로 표현 | 고객, 상품, 주문 |
| 속성(Attribute) | 물리적 개념에서는 칼럼(Column)으로 표현 | 고객아이디, 고객명, 주소 |
| 관계(Releationship) | 기본키와 참조키로 정의 됨(일대일, 일대다) | 고객과 주문과의 관계 |

데이터 모델링

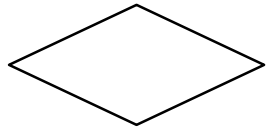
◆ 개념적 설계

현실세계를 추상화(특성화)하여 개체 타입과 관계를 파악하여 표현하는 과정

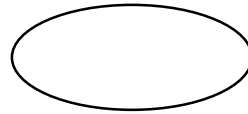
→ 개체 관계도(E-R 다이어그램) : Entity-Relationship Diagram



개체



관계



속성

◆ 논리적 설계

개념적 설계에서 만들어진 구조를 논리적으로 구현 가능한 데이터 모델로 변환하는 단계로 사용자가 알아볼 수 있는 형태로 변환하는 과정 -> 테이블(표) 형태

◆ 물리적 설계

논리적 데이터베이스 구조를 실제 기계가 처리하기에 알맞도록 내부 저장 장치 구조와 접근 경로 등을 설계하는 과정

예) name char(20) – name은 문자형 20Byte를 의미함

데이터 베이스 설계

현실 단계



개념 단계

이름

전화번호

주소

회원

논리 단계

회원

| | | |
|----|------|----|
| 이름 | 전화번호 | 주소 |
|----|------|----|

물리 단계

name CHAR(20)
phone TEXT
address TEXT

개체

특성

값

개체타입

속성

값

레코드타입

특성

값

자료형타입

특성

값

관계형 데이터베이스

관계형 데이터 모델

- 데이터간의 관계에 초점을 둔 모델로 현재 가장 많이 사용하는 모델이다.

예) 회사의 직원정보, 소속된 부서정보

- 직원 정보와 부서 정보를 하나의 묶음으로 관리하면 데이터 구조가 간단해진다. 하지만 같은 부서 직원들은 부서 정보가 중복되므로 효율적인 관리가 어려워진다. 왜냐하면 부서 이름이 바뀌면 직원들의 부서 정보를 일일이 찾아서 수정해주어야 한다.

| 직원 정보 | 직원 번호 | 직원 이름 | 직원 직급 | 부서이름 | 위치 |
|-------|-------|-------|-------|------|----|
| 직원 번호 | 0001 | 홍길동 | 과장 | 회계팀 | 서울 |
| 직원 이름 | 0002 | 성춘향 | 대리 | 연구소 | 수원 |
| 직원 직급 | 0003 | 이몽룡 | 사원 | 영업팀 | 분당 |
| 부서이름 | 0004 | 심청이 | 사원 | 회계팀 | 서울 |
| 위치 | | | | | |

데이터 중복발생

※ 정규화 전의 형태

관계형 데이터베이스

| 부서 정보 |
|-------|
| 부서 코드 |
| 부서 이름 |
| 위치 |

| 부서 코드 | 부서 이름 | 위치 |
|-------|-------|----|
| 10 | 회계팀 | 서울 |
| 20 | 연구소 | 수원 |
| 30 | 영업팀 | 분당 |

| 사원 정보 |
|-------|
| 사원 번호 |
| 사원 이름 |
| 사원 직급 |
| 부서 코드 |

| 사원 번호 | 사원 이름 | 사원 직급 | 부서코드 |
|-------|-------|-------|------|
| 0001 | 홍길동 | 과장 | 10 |
| 0002 | 성춘향 | 대리 | 20 |
| 0003 | 이몽룡 | 사원 | 30 |
| 0004 | 심청이 | 사원 | 10 |

※ 정규화 후의 형태 -> 1대 多의 구조로 변경된다.
한 부서에는 여러 명의 사원이 존재한다.

관계형 데이터베이스

관계형 데이터베이스의 구성 요소

● 테이블(Table)

표 형태의 데이터 저장 공간을 테이블이라고 한다. 2차원 형태로 행과 열로 구성

행(ROW) - 저장하려는 하나의 개체를 구성하는 여러 값을 가로로 늘어뜨린 형태다.

열(COLUMN) - 저장하려는 데이터를 대표하는 이름과 공통 특성을 정의

학생

| 학번 | 이름 | 생년월일 | 학과명 |
|----------|-----|-------------|--------|
| 20150001 | 오상식 | 1987. 6. 10 | 컴퓨터공학과 |
| 20171010 | 최정보 | 1995. 5. 5 | 전자공학과 |
| 20182121 | 김나래 | 1993. 12. 1 | 기계공학과 |

속성, 열, 칼럼, 애트리뷰트

튜플, 레코드, 행

관계형 데이터베이스

관계형 데이터베이스의 구성 요소

● 특별한 의미를 지닌 열 - 키

기본키(Primary Key)

- 테이블의 지정된 행을 식별할 수 있는 유일한 값이어야 한다.
- 값의 중복이 없어야 한다.
- NULL값을 가질 수 없다.
- 주민등록번호, 학번, 사번 등

보조키

- 대체키 또는 후보키라 하며 후보키 중에서 기본키로 지정되지 않은 열이다.

관계형 데이터베이스

외래키(FK : Foreign Key)

- 특정 테이블에 포함되어 있으면서 다른 테이블의 기본키로 지정된 키



SQL이란?

SQL(Structured Query Language)

- '에스큐엘', 또는 '시퀄'이라 부른다.
- 사용자와 데이터베이스 시스템 간에 의사 소통을 하기 위한 언어이다.
- 사용자가 SQL을 이용하여 DB 시스템에 데이터의 검색, 조작, 정의 등을 요구하면 DB 시스템이 필요한 데이터를 가져와서 결과를 알려준다.

| 구분 | 개념 |
|--|----------------------------------|
| DDL(Data Definition Language) - 데이터 정의어 | 테이블을 포함한 여러 객체를 생성, 수정, 삭제하는 명령어 |
| DML(Data Manipulation Language) - 데이터 조작어 | 데이터를 저장, 검색, 수정, 삭제하는 명령어 |
| DCL(Data Control Language) - 데이터 제어어 | 데이터 사용 권한과 관련된 명령어 |

SQL – DDL

DDL(데이터 정의어)

- ▷ 테이블 생성(만들기)
create table 테이블이름(
 name char(10),
 age int
)
- ▷ 테이블 삭제
drop table 테이블 이름
- ▷ 테이블 변경
alter table 테이블 이름 **add** 칼럼추가

SQL – DML

DML(데이터 조작어)

- ▷ 자료 삽입(insert)

insert into 테이블이름(칼럼명) **values** (값1, 값2)

- ▷ 자료 조회(select)

select 칼럼명 **from** 테이블이름

- ▷ 자료 수정(update)

update 테이블이름 **set='변경내용'** **where** 칼럼명

- ▷ 자료 삭제(delete)

delete from 테이블 이름

SQL - DCL

DCL(데이터 제어어)

▷ 커밋과 롤백

트랜잭션(작은 업무 단위) 완료를 의미하는 명령어 - **commit**

변경사항을 취소하고 원래대로 복구하는 명령어 - **rollback**

▷ 권한 부여와 해제

DB 권한을 부여하는 명령어 - **grant**

DB 권한을 해제하는 명령어 - **revoke**

sqlite3

❖ sqlite3

Oracle나 MySql 같은 데이터베이스 관리 시스템이지만, 서버에 위치하지 않고 응용프로그램에 넣어 사용하는 파일(모듈)이다.

Python에서는 내장된 라이브러리로 **import sqlite3**로 사용한다.

❖ DB Browser for sqlite3

오픈소스 소프트웨어로 SQLite 데이터베이스를 GUI 기반으로 편리하게 조작할 수 있도록 해주는 tool이다. 데이터베이스 파일을 생성, 검색 및 편집하려는 사용자를 위해 설계된 고품질의 시각적 오픈 소스 도구입니다

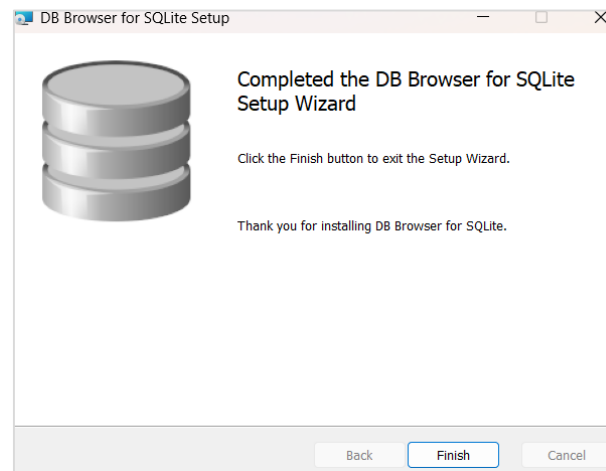
sqlite3

❖ DB 브라우저 설치

Windows

Our latest release (3.13.1) for Windows:

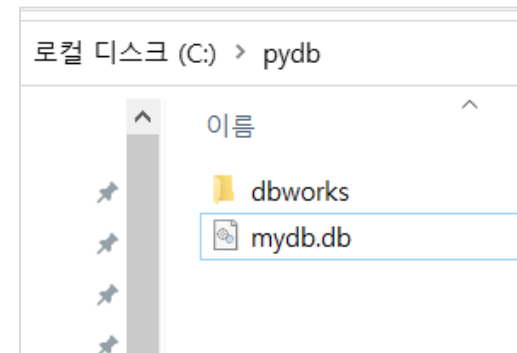
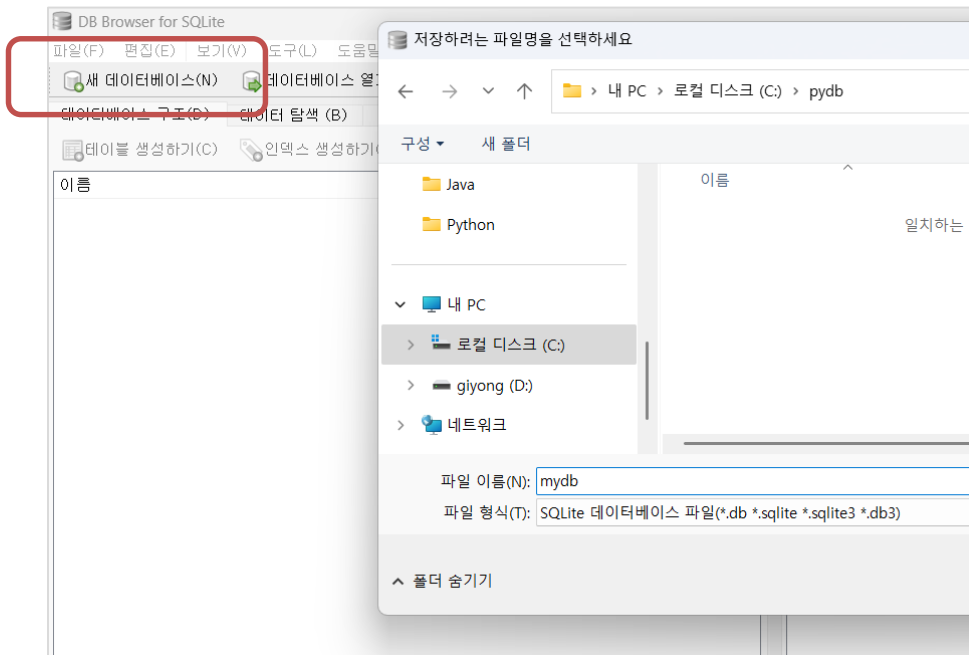
- DB Browser for SQLite - Standard installer for 32-bit Windows
- DB Browser for SQLite - .zip (no installer) for 32-bit Windows
- DB Browser for SQLite - Standard installer for 64-bit Windows
- DB Browser for SQLite - .zip (no installer) for 64-bit Windows



sqlite3


❖ 데이터베이스 생성

새 데이터베이스 > mydb.db 이름으로 저장



sqlite3

Sqlite3 Document > Alphabet.. > CREATE TABLE



Home About Documentation Download

▼ Document Lists And Indexes

- [Alphabetical Listing Of All Documents](#)
- [Website Keyword Index](#)
- [Permuted Title Index](#)

- Overview Documents
- Programming Interfaces
- Extensions
- Features

AS SELECT Statements

"SELECT" statement creates and populates a database table based on the result of a SELECT statement. The name of each column is the same as the name of the column in the SELECT statement. The affinity of each column is determined by the [expression affinity](#) of the corresponding expression in the SELECT statement.

| Expression Affinity |
|---------------------|
| TEXT |
| NUMERIC |
| INTEGER |
| REAL |
| BLOB (a.k.a "NONE") |

sqlite3

■ 데이터 타입 및 기초 문법

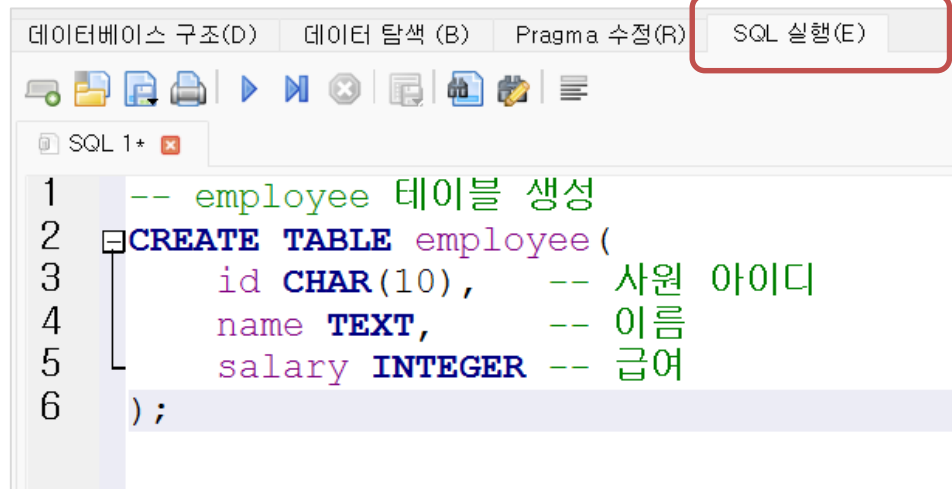
| 데이터 타입 | 설명 |
|-----------------|----------------|
| char | 고정길이 문자 |
| text | 가변길이 문자(주로 사용) |
| integer | 정수값 |
| real | 실수값 |
| datetime | 날짜와 시간 |

- 문자는 홑따옴표만 사용할 수 있음
- 한줄 주석 처리는 (--), 여러 줄 주석은 /* ~ */
- 문장의 끝은 세미콜론(;) 사용

DDL – 테이블 생성

■ 테이블 생성

```
CREATE TABLE 테이블이름(  
    컬럼명 자료형  
)
```



```
1  -- employee 테이블 생성  
2  CREATE TABLE employee(  
3      id CHAR(10),    -- 사원 아이디  
4      name TEXT,      -- 이름  
5      salary INTEGER -- 급여  
6  );
```

SQL 실행후
코드 작성

DML – 자료 삽입

- 자료 삽입(Insert)

INSERT INTO 테이블이름(칼럼명) **VALUES** (값1, 값2)

```
-- 데이터 검색
SELECT * FROM employee;

-- 데이터 삽입
INSERT INTO employee (id, name, salary) VALUES ('e101', '이정후', 3000000);
INSERT INTO employee (id, name, salary) VALUES ('e102', '신유빈', 2500000);

COMMIT; -- 커밋 (변경사항 저장)
```

| | id | name | salary |
|---|------|------|---------|
| 1 | e101 | 이정후 | 3000000 |
| 2 | e102 | 신유빈 | 2500000 |

메러 없이 실행 완료.
결과: 8ms의 시간이 걸려서 2 행이 반환되었습니다
9번째 줄:
SELECT * FROM employee;

DML – 자료 검색

▷ 자료 조회(Select)

SELECT 컬럼명 **FROM** 테이블이름 (전체 자료 조회)

SELECT 컬럼명 **FROM** 테이블이름 **WHERE** 컬럼명 = 값 (특정 자료 조회)

```
-- id가 'e101'인 사원의 이름과 급여 정보 출력
SELECT name, salary FROM employee;
SELECT name, salary FROM employee WHERE id = 'e101';

-- 이름이 신유빈인 사원의 급여 정보 출력
SELECT name, salary FROM employee WHERE name LIKE '신유빈';
```

DML – 정렬

▷ 자료 조회(select) - 정렬하기

SELECT 컬럼명 **FROM** 테이블이름 **ORDER BY** 컬럼명 **DESC**

```
-- 나이가 적은 순으로 정렬하기 (오름차순) --  
SELECT * FROM employee ORDER BY age;  
  
-- 급여가 높은 순으로 정렬하기 (내림차순) --  
SELECT * FROM employee ORDER BY salary DESC;
```

| | emp_id | name | age | salary |
|---|--------|------|-----|--------|
| 1 | e103 | 안산 | 21 | NULL |
| 2 | e102 | 박인비 | 31 | 20000 |
| 3 | e101 | 추신수 | 39 | 10000 |

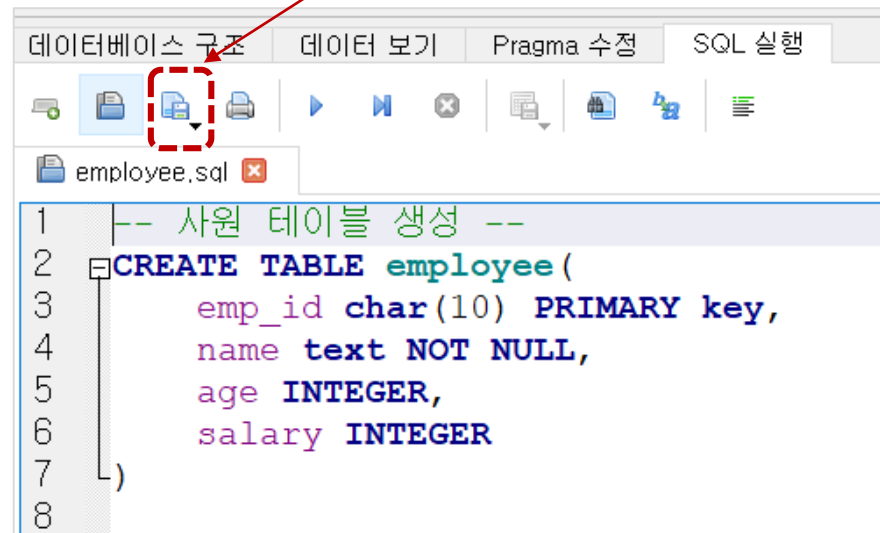
| | emp_id | name | age | salary |
|---|--------|------|-----|--------|
| 1 | e102 | 박인비 | 31 | 20000 |
| 2 | e101 | 추신수 | 39 | 10000 |
| 3 | e103 | 안산 | 21 | NULL |

DB Browser

▷ sql 파일로 저장하기

sql 실행 탭 > sql 파일로 저장하기

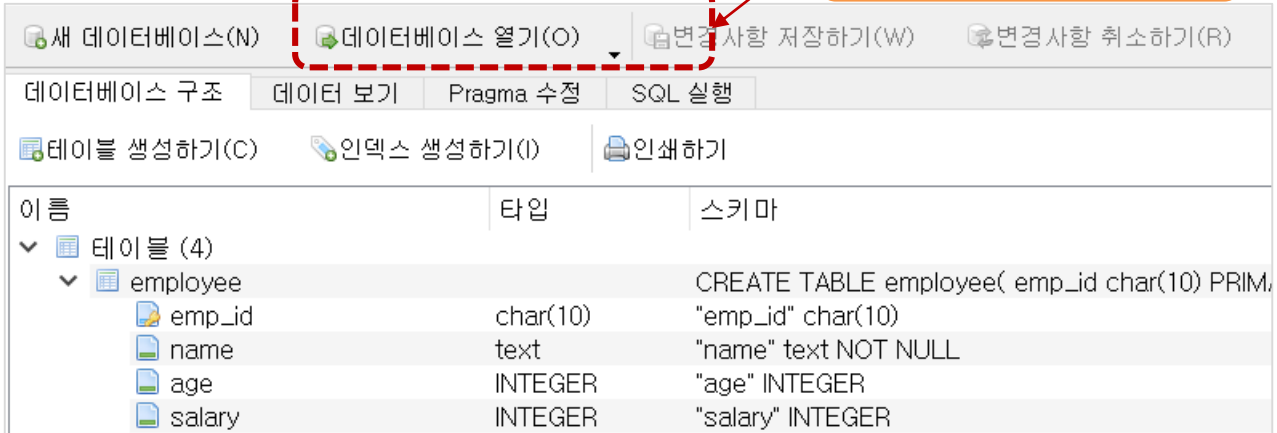
SQL 파일 저장



DB Browser

■ 테이블 생성 확인

① mydb 연결하기



The screenshot shows the DB Browser for SQLite interface. The 'Database Open' button is highlighted with a red dashed box. The interface also shows the 'Database Structure' tab selected, displaying the 'employee' table structure.

| 이름 | 타입 | 스키마 |
|----------|----------|--|
| 테이블 (4) | | |
| employee | | CREATE TABLE employee(emp_id char(10) PRIM, |
| emp_id | char(10) | "emp_id" char(10) |
| name | text | "name" text NOT NULL |
| age | INTEGER | "age" INTEGER |
| salary | INTEGER | "salary" INTEGER |

DML – 자료 수정

- 자료 수정

UPDATE 테이블명 **SET** 컬럼명=값 **WHERE** 컬럼명=값

```
-- 이정후의 급여를 400만원으로 변경  
UPDATE employee SET salary = 4000000 WHERE name LIKE '이정후';  
  
-- id가 'e102'인 사원의 이름을 '임시현'으로 변경  
UPDATE employee SET name = '임시현' WHERE id = 'e102';
```

| | id | name | salary | |
|---|------|------|---------|--|
| 1 | e101 | 이정후 | 4000000 | |
| 2 | e102 | 임시현 | 2500000 | |

DML – 자료 삭제

- 자료 삭제

DELETE FROM 테이블명 – 전체 자료 삭제

DELETE FROM 테이블명 **WHERE** 컬럼명=값 – 특정 자료 삭제

| | id | name | salary |
|---|------|------|---------|
| 1 | e101 | 이정후 | 3000000 |
| 2 | e102 | 신유빈 | 2500000 |
| 3 | e103 | 테스터 | NULL |

```
-- 이름이 '테스터'인 사원의 정보 출력  
DELETE FROM employee WHERE name = '테스터';
```

무결성 제약

◆ 제약조건

테이블들은 각 속성(칼럼)에 대한 무결성을 유지하기 위한 다양한 제약 조건 (Constraints)이 적용되어 있다.

제약 조건에는 NOT NULL, 기본키, 외래키, CHECK 등이 있다.

✓ NOT NULL

칼럼을 정의할 때 NOT NULL 제약 조건을 명시하면 반드시 데이터를 입력해야 한다.

칼럼명 데이터 타입 **NOT NULL**

무결성 제약

◆ 제약조건

✓ 기본키(PRIMARY KEY)

기본키는 Primary Key라고도 하며, UNIQUE와 NOT NULL 속성을 동시에 가진 제약 조건으로 테이블 당 1개의 기본키만 생성할 수 있다.

칼럼명 데이터 타입 **PRIMARY KEY**

또는

PRIMARY KEY(칼럼명, ...)

DDL – 테이블 삭제

- 테이블 삭제

DROP TABLE 테이블명

```
-- 테이블 삭제  
DROP TABLE employee;
```

무결성 제약

◆ 제약조건

- ✓ id 중복저장 문제 발생 : 기본키 설정
- ✓ name : 비어있지 않도록 설정 – NOT NULL

```
-- employee 테이블 생성
CREATE TABLE employee_new(
    id CHAR(10) PRIMARY KEY,    -- 기본키
    name TEXT NOT NULL,         -- 이름
    salary INTEGER               -- 급여
);
```

무결성 제약

◆ 제약조건

```
-- 자료 삽입
INSERT INTO employee_new (id, name, salary) VALUES ('e101', '이정후', 3000000);
INSERT INTO employee_new (id, name, salary) VALUES ('e102', '신유빈', 2500000);
-- 아이디 중복 - 기본키 위배로 저장 안됨
INSERT INTO employee_new (id, name, salary) VALUES ('e102', '임시현', 3500000);
-- 이름 - NOT NULL 제약조거 위배
INSERT INTO employee_new (id, salary) VALUES ('e103', 2500000);
```

| | id | name | salary |
|---|------|------|---------|
| 1 | e101 | 이정후 | 3000000 |
| 2 | e102 | 신유빈 | 2500000 |

파이썬으로 DB 관리

➤ DB 처리 프로세스

1. 데이터베이스 연결 객체 생성

```
conn = sqlite3.connect("c:/pydb/mydb.db")
```

2. cursor 객체 생성 – DB 테이블 작업 객체

```
cur = conn.cursor()
```

3. execute() 함수 실행

```
cur.execute(sql)
```

DB 연결

➤ 파이썬으로 sqlite3에 연결

```
import sqlite3

# db 연결 확인
try:
    conn = sqlite3.connect("c:/pydb/mydb.db")
    print("DB 연결 및 실행 성공")
except sqlite3.Error as e:
    print(f"DB 오류 발생: {e}")
```

DB – 사원 관리

➤ 전체 검색

```
def select():  
    # with ~ as 구문: close() 사용 안함  
    with sqlite3.connect("c:/pydb/mydb.db") as conn: #db 연결  
        cur = conn.cursor() # 작업 객체 생성  
        sql = "SELECT * FROM employee_new"  
        cur.execute(sql) # sql 실행  
  
        rs = cur.fetchall() #검색된 자료 모두 가져오기  
        # print(rs)  
        for i in rs:  
            print(i)  
  
# 사원 검색  
select()
```

```
('e101', '이정후', 3000000)  
( 'e102', '신유빈', 2500000)
```

DB – 사원 관리

➤ 사원 추가

```
def insert():  
    with sqlite3.connect("c:/pydb/mydb.db") as conn:  
        cur = conn.cursor()  
        sql = "INSERT INTO employee_new VALUES(?, ?, ?)"  
        cur.execute(sql, ('e103', '최민정', 3500000))  
        conn.commit() #커밋 실행  
        print("회원 추가 완료!")  
  
# 사원 추가  
insert()
```


DB – 사원 관리

➤ 특정 사원 1명 검색

```
def select_one():  
    with sqlite3.connect("c:/pydb/mydb.db") as conn: #db 연결  
        cur = conn.cursor() # 작업 객체 생성  
        sql = "SELECT * FROM employee_new WHERE id = 'e102'"  
        cur.execute(sql) # sql 실행  
  
        rs = cur.fetchone() #검색된 자료 1건 가져옴  
        print(rs)  
  
# 특정 사원 검색  
select_one()
```

DB – 사원 관리

➤ 사원 정보 수정

```
def update():  
    with sqlite3.connect("c:/pydb/mydb.db") as conn:  
        cur = conn.cursor()  
        sql = "UPDATE employee_new SET salary = ? WHERE name = ?"  
        cur.execute(sql, (5000000, '이정후'))  
        conn.commit() #커밋 실행  
        print("회원 수정 완료!")
```

```
# 사원 정보 수정  
update()
```

```
회원 수정 완료!  
( 'e101', '이정후', 5000000)  
( 'e102', '신유빈', 2500000)  
( 'e103', '최민정', 3500000)
```

DB – 사원 관리

➤ 사원 정보 삭제

```
def delete():  
    with sqlite3.connect("c:/pydb/mydb.db") as conn:  
        cur = conn.cursor()  
        sql = "DELETE FROM employee_new WHERE id = ?"  
        cur.execute(sql, ('e102', ))  
        conn.commit() #커밋 실행  
        print("회원 수정 완료!")
```

사원 정보 삭제

```
delete()
```

DB – 회원 관리

- 데이터 베이스 생성 – my.db
 - 회원 테이블 생성 - member

| 칼럼 이름 | 자료형 | 설명 |
|----------|---------------------|------|
| memberId | CHAR(5) PRIMARY KEY | 회원번호 |
| passwd | CHAR(10) NOT NULL | 비밀번호 |
| name | TEXT NOT NULL | 이름 |
| gender | CHAR(4) | 성별 |
| age | INTEGER | 나이 |

DB – 회원 관리

- 함수 정의 및 호출

```
#create_table()  
#insert_member()  
#select_one()  
#update_member()  
#delete_member()  
select_member()
```

- DB 접속 함수 : getconn() 정의

```
import sqlite3  
  
def getconn():  
    # DB 접속함수 정의  
    conn = sqlite3.connect("c:/pydb/will.db")  
    return conn
```

DB – 회원 관리

- 테이블 생성 – 테이블 이름 : member, def create_table()

```
def create_table():  
    # 테이블 생성  
    conn = getconn()  
    cur = conn.cursor()  
    sql = """  
    CREATE TABLE member(  
        memberId CHAR(5) PRIMARY KEY,  
        passwd CHAR(10) NOT NULL,  
        name TEXT NOT NULL,  
        gender CHAR(4),  
        age INTEGER  
    )  
    """  
    cur.execute(sql)  
    conn.commit()  
    print("테이블 생성")  
    conn.close()
```

DB – 회원 관리

- 자료 삽입

- def insert_member() : 동적 바인딩 방식

```
def insert_member():  
    # 자료 삽입  
    conn = getconn()  
    cur = conn.cursor()  
    # 자료 삽입 방법 - 동적 바인딩('? ' 기호로 대응)  
    sql = "INSERT INTO member VALUES (?, ?, ?, ?, ?)"  
    cur.execute(sql, ('10001', 'm123456789', '지민', '남자', 30)) # 튜플  
    conn.commit()  
    print("회원 추가")  
    conn.close()
```

테이블(T): member

| | memberId | passwd | name | gender | age |
|---|----------|------------|------|--------|-----|
| | 필터 | 필터 | 필터 | 필터 | 필터 |
| 1 | 10001 | m123456789 | 지민 | 남자 | 27 |
| 2 | 10002 | m123456780 | 민정 | 여자 | 24 |
| 3 | 10003 | m123456781 | RM | 남자 | 28 |

DB – 회원 관리

● 자료 전체 검색

- def select_member() > fetchall() 함수로 자료를 반환 받음
- 자료 검색 시엔 conn.commit()은 사용 안함

```
def select_member():  
    # 자료 검색  
    conn = getconn()  
    cur = conn.cursor()  
    sql = "SELECT * FROM member"  
    cur.execute(sql)  
    rs = cur.fetchall()  
    for i in rs:  
        print(i)  
    conn.close()
```

| | | | | |
|-----------|---------------|-------|-------|-----|
| ('10001', | 'm123456789', | '지민', | '남자', | 30) |
| ('10002', | 'm123456780', | '민정', | '여자', | 24) |
| ('10003', | 'm123456781', | 'RM', | '남자', | 28) |

DB – 회원 관리

- 자료 1개 검색

- def select_one() > fetchone() 함수로 자료 반환

```
def select_one():  
    # 특정한 자료 1개 검색  
    conn = getconn()  
    cur = conn.cursor()  
    sql = "SELECT * FROM member WHERE memberId = ?"  
    cur.execute(sql, ('10002',)) # 튜플 - 자료 1개(콤마 붙임)  
    rs = cur.fetchone()  
    print(rs)  
    conn.close()
```

```
('10002', 'm123456780', '민정', '여자', 24)
```

DB – 회원 관리

- 자료 수정

- def update_member()

```
def update_member():  
    # 자료 수정  
    conn = getconn()  
    cur = conn.cursor()  
    sql = "UPDATE member SET age = ? WHERE name = ?"  
    cur.execute(sql, (27, '지민'))  
    conn.commit()  
    conn.close()
```

```
('10001', 'm123456789', '지민', '남자', 27)
```

DB – 회원 관리

- 자료 삭제

- def delete_member()

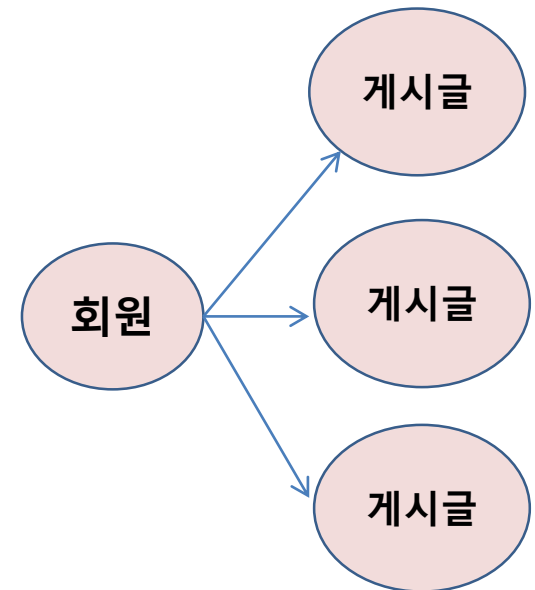
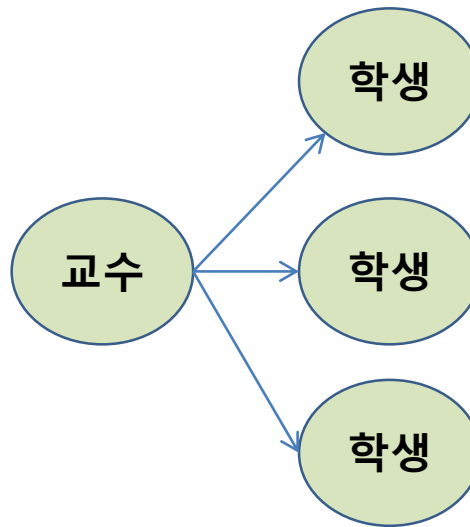
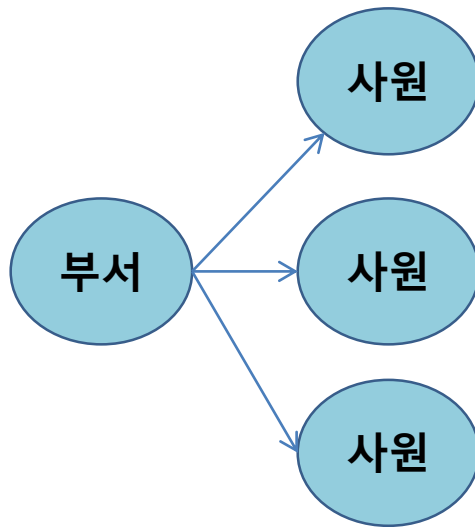
```
def delete_member():  
    # 자료 삭제  
    conn = getconn()  
    cur = conn.cursor()  
    sql = "DELETE FROM member WHERE memberId = ?"  
    cur.execute(sql, (10003, ))  
    conn.commit()  
    conn.close()
```

```
('10001', 'm123456789', '지민', '남자', 27)  
( '10002', 'm123456780', '민정', '여자', 24)
```

관계(Releation)

엔티티(Entity) 관계(releation)

- 1대 多의 관계, 1대 1관계, 多 대 多 관계



관계(Releation)

외래키(FK : Foreign Key)

- 특정 테이블에 포함되어 있으면서 다른 테이블의 기본키로 지정된 키



관계(Releation) – 외래키

부서와 직원 테이블 생성

- 사원(employee) 테이블에 Foreign Key 설정

```
-- 부서 테이블 --
CREATE TABLE department(
    deptid INTEGER,
    deptname TEXT NOT NULL,
    location TEXT NOT NULL,
    PRIMARY KEY(deptid)
);

-- 사원 테이블 --
CREATE TABLE employee(
    empid INTEGER,
    name TEXT NOT NULL,
    age INTEGER,
    deptid INTEGER,
    PRIMARY KEY(empid),
    FOREIGN KEY(deptid) REFERENCES department(deptid)
);
```

관계(Releation) – 외래키

부서와 직원 자료 추가

```
-- 부서 자료 추가 --  
INSERT INTO department VALUES (10, '전산팀', '서울' )  
INSERT INTO department VALUES (20, '총무팀', '인천' )  
  
-- 사원 자료 추가 --  
INSERT INTO employee VALUES (501, '이강', 23, 10)  
INSERT INTO employee VALUES (502, '김산', 24, 20)  
INSERT INTO employee VALUES (503, '북한강', 25, 30)
```

에러가 발생하여 실행 중단됨.

결과: FOREIGN KEY constraint failed

19번째 줄:

```
INSERT INTO employee VALUES (503, '북한강', 25, 30)
```

| | deptid | deptname | location |
|---|--------|----------|----------|
| | 필터 | 필터 | 필터 |
| 1 | 10 | 전산팀 | 서울 |
| 2 | 20 | 총무팀 | 인천 |

| empid | name | age | deptid |
|-------|------|-----|--------|
| 필터 | 필터 | 필터 | 필터 |
| 501 | 이강 | 23 | 10 |
| 502 | 김산 | 24 | 20 |

부서 테이블에 부서코드 30이 없으므로, 사원 503을 추가 할 수 없다.

관계(Releation) – 외래키

부서 자료 삭제

```
-- 부서 자료 삭제 --  
DELETE FROM department WHERE deptid = 20
```

에러가 발생하여 실행 중단됨.
결과: FOREIGN KEY constraint failed
17번째 줄:
DELETE FROM department WHERE deptid = 20

사원 테이블에서 부서 테이블을 참조하고 있으므로 삭제할 수 없다.

DB – book 관리

- 데이터 베이스 생성 – test.db
 - 책 테이블 생성 - book

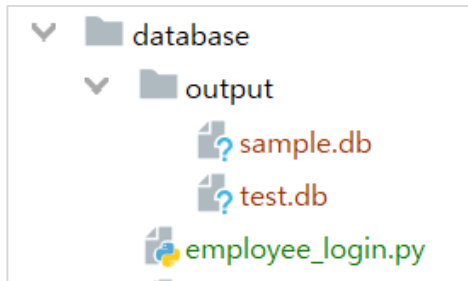
| 칼럼 이름 | 자료형 | 설명 |
|-----------|--------------------------------------|------|
| book_no | INTEGER PRIMARY KEY AUTOINCREMENT | 책번호 |
| title | TEXT NOT NULL | 책제목 |
| publisher | TEXT NOT NULL | 저자 |
| page | INTEGER | 페이지수 |

DB – book 관리

- db생성 및 connect(연결) – getconn() 정의

```
import sqlite3

def getconn(): # 데이터베이스 연결
    conn = sqlite3.connect('./output/test.db')
    return conn
```



DB – book 관리

- book 테이블 생성하기

```
def create_table(): # book 테이블 생성
    conn = getconn()
    cur = conn.cursor()
    sql = """
    CREATE TABLE book(
        book_no INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        publisher TEXT NOT NULL,
        page INTEGER
    )
    """
    cur.execute(sql)
    print("book 테이블 생성")
    conn.commit()
    conn.close()
```

DB – book 관리

- book – 책 추가하기

```
def insert_book():    # 책 추가 입력
    conn = getconn()
    cur = conn.cursor()
    sql = "INSERT INTO book(title, publisher, page) VALUES (?, ?, ?)"
    cur.execute(sql, ('웹 표준의 정석', '고경희', 600))
    conn.commit()
    conn.close()
```

sequence(순서) -> autoincrement(1씩 자동증가)

book_no는 수동 입력하지 않아야 함

DB – book 관리

- book – 자료 전체 목록

```
def select_book(): # 책 목록 검색
    conn = getconn()
    cur = conn.cursor()
    sql = "SELECT * FROM book"
    cur.execute(sql)
    rs = cur.fetchall() # rs=resultSet의 약자
    for i in rs:
        print(i)
    conn.close()
```

```
(1, '웹 표준의 정석', '고경희', 600)
(2, '점프 투 파이썬', '박응용', 350)
```

DB – book 관리

- book – 책 1권 검색하기

```
def select_one(): # 특정한 책 검색
    conn = getconn()
    cur = conn.cursor()
    sql = "SELECT * FROM book WHERE book_no=?"
    cur.execute(sql, (2,)) # 튜플은 1개일때 코머를 꼭 붙인다.
    rs = cur.fetchone()
    print(rs)
    conn.close()
```

DB – book 관리

- book – 책 정보 변경 및 삭제하기

```
def update_book(): # 책 정보 변경
    conn = getconn()
    cur = conn.cursor()
    # 1번 책의 페이지를 600 -> 650으로 변경
    sql = "UPDATE book SET page=? WHERE book_no=?"
    cur.execute(sql, (650, 1))
    conn.commit()
    conn.close()

def delete_book(): # 책 삭제
    conn = getconn()
    cur = conn.cursor()
    # 2번책 삭제
    sql = "DELETE FROM book WHERE book_no=?"
    cur.execute(sql, (2,))
    conn.commit()
    conn.close()
```