데이터베이스 구축 및
Python, SQLite, Tkinter를 조합한
데이터 검색 프로그램 제작

- 활용 데이터베이스 구축 및
실질적 활용 가능한 데이터 검색 프로그램 제작 -

과 목 명 : 데이터베이스
성　　명 : 이　준　혁
소　　속 : 컴퓨터공학과
학　　번 : 7512321
지도교수 : 김송현 교수님

제 출 일 : 2020. 12. 14(월)

목차

1. 서론

1.1. 프로젝트의 목적

지나온 두 개의 과제들은 지금부터 서술할 프로젝트의 준비과정이었다. 최종 장이 되는 프로젝트의 목적은 '데이터베이스의 개념을 이해하여 데이터베이스를 구축하고, 데이터베이스에 검색, 입력, 수정, 삭제 명령어를 적절히 사용할 수 있으며, 이를 위하여 GUI를 사용하여 응용프로그램을 개발할 수 있는 능력을 구비한다.'라고 명시되어 있다. 첫 과제에서 우리가 사용할 도구들에 대해 알아보았고 그 다음 과제에서 그 도구들을 직접 사용해보며 GUI에 대한 느낌을 다잡았다.

주어진 목적에 대해서 생각해보면 중요한 것은 두 가지라는 것을 알 수 있다. 첫째는 데이터베이스가 무엇인지 이해하고 직접 구축하는 것이다. 지난 과제는 이미 기존에 있는 샘플 데이터를 활용하였기 때문에 개발에 집중했지만 이번에는 이론 시간에 배운 데이터베이스 자체에 대한 실제 적용과 활용이 중요하다. 이 부분에 대해서 교수님께서 의미가 있는 데이터를 선택하면 좋겠다는 말씀을 하셨기에 참고를 하면 될 것 같다. 두 번째는 GUI 자체이다. 앞선 과제로 GUI가 어떤 방향성을 가져야 하는지에 대해 고민을 해보며 느낌을 다잡긴 했지만 그건 느낌 정도일 뿐이다. 실제 GUI는 완전히 사용자에 친화된 인터페이스로, 사용자의 모든 행동과 예외 상황을 고려해서 한 치의 오류도 일으키지 않은 채 사용자의 요구사항 전체를 수행할 수 있어야 할 것이다.

그럼 이제 교수님이 말씀하신 의미 있는 데이터에 대해 생각해보자. 이것에 대해서는 하나의 스토리가 나오면 좋겠다는 말씀을 덧붙이셨다. 사실 데이터베이스는 물론 무언가를 만들고 개발한다는 것은 기존의 것이 불편하거나 원래의 방식으로는 해결할 수 없는 것을 타파하기 위한 행동이다. 그래서 우리 주변의 불편한 문제들 중 데이터베이스를 활용하여 해결할 수 있는 문제를 찾아 프로젝트를 진행해보기로 했다.

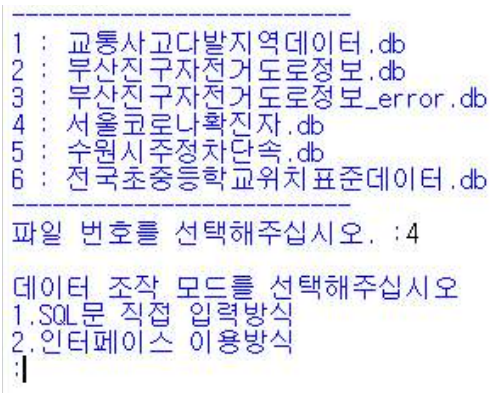
또 말씀하신 것은 인터페이스 제작에 데이터의 형태가 참고되었으면 한다고도 하셨다. 다만 이 부분은 고민을 해보았는데, 사실 GUI라는 것은 그래픽을 이용한 것인데 필자는 디자인적 감각이 심히 부족하기 때문이다. 그래서 다른 것들과 차별적인 특수한 디자인 설계를 가지기가 힘들 것 같았기에 범용성 높은 디자인과 고민을 하였다. 그래서 결정내린 것은 특수한 데이터베이스에만 사용할 수 있는 설계보다 어느 데이터베이스를 들고와도 활용할 수 있는, 마치 수업시간에 실습용으로 사용했던 'DB Browser' 같은 형식의 디자인을 하기로 선택했다.

1.2. 앞선 과제의 정리

제일 먼저 했던 과제1에서는 프로젝트의 준비의 역할이 컸다. 최종 프로그램 개발을 위해 필요한 SQLite와 Python, Tkinter에 대한 조사가 중점이 되었다. 우리가 사용할 각각의 도구들에 대해 조사하여 SQLite는 데이터베이스를 조작하고 다루는

데에 필요한 것임을, Tkinter는 GUI설계를 위한 파이썬의 보조임을, Python은 이 모두를 조화롭게 엮어 원하는 대로 프로그램을 작성하고 설계할 수 있는 그림판과도 같은 것임을 알았고 활용하는 방법도 간략하게나마 학습했다.

과제2는 프로젝트를 본격적으로 하기에 앞서 연습을 해보는 역할이었다. 과제1에서 조사한 각 도구들이 실제로 작동하는지, 어떻게 작동하는지, 활용을 하려면 어떻게 해야할지 등을 학습했다. 그를 토대로 샘플 데이터를 가지고 프로젝트의 견본같은 것도 만들어 보았다.



<과제2 수행 결과물>

결과적으로 사용자 인터페이스 설계에 대한 철학이나 취지를 이해하고 프로그램의 설계에 대해 생각해보게 되었다. 이제는 한학기동안 배워온 것들을 모두 활용하여 데이터베이스의 설계부터 그 데이터를 다룰 프로그램을 사용자에게 친숙한 형태로 만들어내야 할 때이다.

2. 본론

2.1. 데이터베이스 구축

2.1.1. 데이터 선정이유

필자가 선택한 것은 생도대를 기반으로 한 데이터베이스이다. 현재의 생도대는 인트라넷을 활용하거나 카톡을 이용하여 각종 신청 혹은 조사사항을 수합하고 있다. 언뜻 보면 체계 또는 시스템을 활용하고 있는 것처럼 보이지만 실상은 사람의 수작업으로 행해지고 있다. 필자가 행정보좌관을 역임하고 현재 군수보좌관을 담당하고 있는 동기의 말을 들어보면 이렇게나 무식하게? 라고 할 정도로 일처리가 이루어지고 있다. 외출신청을 받는 경우를 보면 인트라넷을 통해 신청하여 수합은 전산을 통해 이루어지지만, 이 데이터를 다룰 수 있는 프로그램, 시스템이 하나도 없기 때문에 신청에 대한 수정이나 삭제, 추가는 물론 검색까지도 모두 사람의 손으로 하고 있는 실정이다.

군수계열의 상황은 더 심각하다. 이쪽은 신청을 위한 체계조차 없어 군수품에 대

한 신청, 관리, 수령 여부 등을 모두 사람의 손으로 이뤄지고 있다. 예를 들어 생활실에 있는 가구들은 모두 군수계열에서 담당하는데, 이런 가구의 현황에 대해 알기 위해서는 각 생활실에 거주하는 생도들이 카카오톡, 웹메일 등을 활용하여 가구 현황을 보내면, 군수계열 종사자들이 이것들을 일일이 하나의 파일로 합치고, 분류하며 누락에 대한 확인조차도 직접 수행한다.

이러한 실정이기에 생도대에서 이뤄지는 외출, 물품 관리, 식사 관리 등을 대상으로 데이터베이스를 만들고자 한다. 지금까지 업무에 실제로 종사하고 있진 않으며 말단 직을 맡았기에 상부에서 어떻게 처리하는지 자세히 아는 바가 없지만 이러한 데이터들을 다룬다면 어떻게 데이터베이스를 만드는게 좋을지부터 시작해서 활용 프로그램 제작까지 이어나가겠다.

2.1.2. 개념적 설계

데이터베이스 설계는 수업 중에 배운 순서를 따라가며 단계적으로 설계해보겠다. 우선으로 데이터베이스에 필요한 부분들을 문장으로 나열하였다.

- 공군사관학교 생도는 교번, 이름, 나이, 성별, 중대, 학년, 학과를 갖는다.
- 생도는 교번으로 식별한다.
- 외출은 식별번호, 외출종류, 귀향/귀영, 일자, 시각, 배차에 대한 정보를 갖는다.
- 외출은 식별번호로 식별한다.
- 한 생도가 여러 외출을 신청할 수 있다.
- 외출을 신청할 때 외출자, 행선지, 최종결재자, 확인관에 대한 정보를 제공한다.
- 각 생도는 식사를 가지며 식사는 식별번호, 식사구분, 날짜, 메뉴에 대한 정보를 갖는다.
- 식사를 신청할 때 신청자, 취식/결식, 사유에 대한 정보를 제공한다.
- 생도는 식사들에 대해 취식이나 결식을 신청할 수 있다.
- 생도는 외출 시 물품을 반입/반출할 수 있다.
- 반입 물품은 관리번호, 물품명, 반입일, 반출예정일에 대한 정보를 갖는다.
- 생도는 반입한 물품을 관리하며, 여러 물품을 반입할 수 있다.
- 반입/반출을 신청할 때 관리자, 사유, 인가자에 대한 정보를 제공한다.

<데이터베이스 설계를 위한 요구사항들>

데이터 선정 이유에 맞게 데이터베이스를 사용하고자 할 때 어떤 용도를 가지고 있는지 명확히 파악해야 한다. 데이터베이스를 사용하여 실제 업무를 처리하는 사용자에게 필요한 데이터의 종류와 처리 방법과 같은 다양한 요구 사항을 수집하는 것이 제일 좋겠으나 지금의 경우에서는 필자의 경험과 주변에서 들려오는 불편함을 대상으로 하여 요구사항을 만들어보았다.

이제 요구사항들을 토대로 개념적 설계를 해야한다. 개념적 설계는 개념적 데이터 모델을 통해 사용자의 요구사항을 표현해야하는데, 이를 위해 E-R 모델을 이용하여 개념적 모델링을 하고자 한다. 먼저 E-R 모델의 핵심요소인 개체를 추출하고, 각 개체의 주요 속성과 키 속성을 선별한다. 다음으로 개체 간의 관계를 결정해야 하고

이 모든 작업이 완료되면 그 결과를 E-R 다이어그램으로 표현해야한다.
다음은 요구사항에서 개체와 속성들을 빨간 색으로 표현해보았다.

○ 공군사관학교 생도는 교번, 이름, 나이, 성별, 중대, 학년, 학과를 갖는다.
○ 생도는 교번으로 식별한다.
○ 외출은 식별번호, 외출종류, 귀향/귀영, 일자, 시각, 배차에 대한 정보를 갖는다.
○ 외출은 식별번호로 식별한다.
○ 한 생도가 여러 외출을 신청할 수 있다.
○ 외출을 신청할 때 외출자, 행선지, 최종결재자, 확인관에 대한 정보를 제공한다.
○ 각 생도는 식사를 가지며 식사는 식별번호, 식사구분, 날짜, 메뉴에 대한 정보를 갖는다.
○ 식사를 신청할 때 신청자, 취식/결식, 사유에 대한 정보를 제공한다.
○ 생도는 식사들에 대해 취식이나 결식을 신청할 수 있다.
○ 생도는 외출 시 물품을 반입/반출할 수 있다.
○ 반입 물품은 관리번호, 물품명, 반입일, 반출예정일에 대한 정보를 갖는다.
○ 생도는 반입한 물품을 관리하며, 여러 물품을 반입할 수 있다.
○ 반입/반출을 신청할 때 관리자, 사유, 인가자에 대한 정보를 제공한다.

<요구사항들에서 개체와 속성을 추출한 것>

이들을 개체와 개체에 속하는 속성들로 구분하여 정리하면 다음과 같다. 왼쪽 열이 개체이고 오른쪽 열이 해당하는 속성들이다. 밑줄 친 속성은 키 속성이다.

생도	교번, 이름, 나이, 성별, 중대, 학년, 학과
외출	식별번호, 외출종류, 귀향/귀영, 일자. 시각, 배차
식사	식별번호, 식사구분, 날짜, 메뉴
물품	관리번호, 물품명, 반입일, 반출예정일

<추출한 개체와 속성을 정리한 결과>

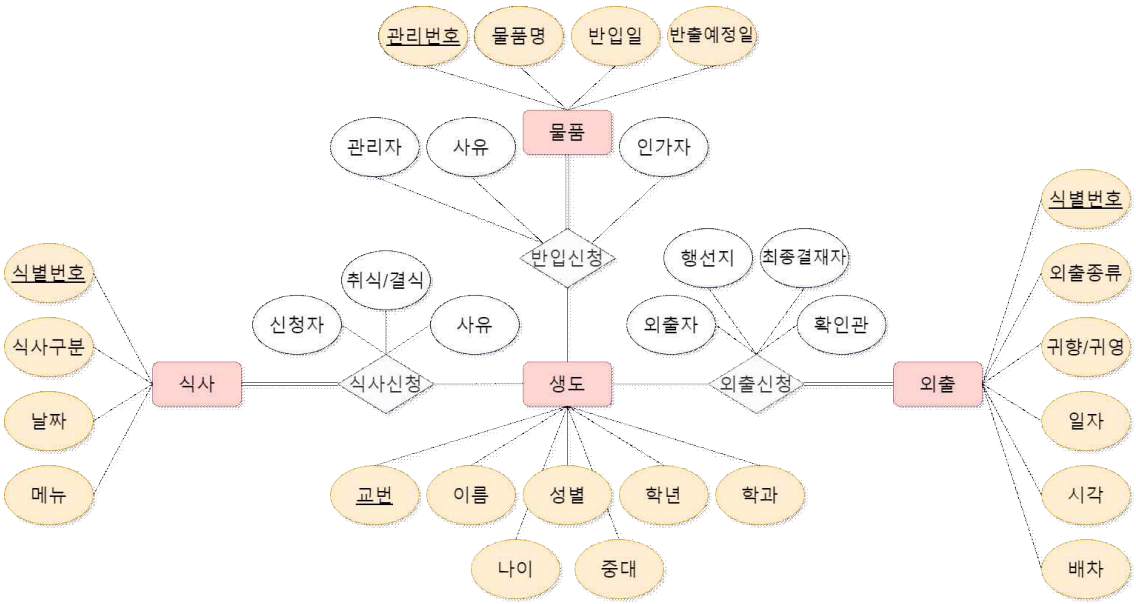
개체와 속성을 추출하였으니 개체 간의 관계를 결정하여야 한다. 관계는 개체 간의 의미있는 연관성으로, 우리의 요구사항에서는 외출신청, 식사신청, 반입신청 세 가지로 결정되는데, 이에 대한 매핑 카디널리티을 결정해야한다. 매핑 카디널리티는 관계를맺고 있는 두 개체에서, 각 개체 인스턴스가 관계를 맺고 있는 상대 개체의 개체 인스턴스의 개수를 의미하는 것으로[1], 일대일, 일대다, 다대다 관계를 말한다.

외출신청	생도 외출	일대다	외출자, 행선지, 최종결재자, 확인관
식사신청	생도 식사	일대다	신청자, 취식/결식, 사유
반입신청	생도 물품	일대다	관리자, 사유, 인가자

<요구사항에서 추출한 관계와 관계 속성>

1) 한빛아카데미, 「데이터베이스 개론 2판」, 김현희, 312쪽.

주어진 요구사항에서 추출한 개체, 속성, 관계 들을 바탕으로 E-R 다이어그램을 표현하면 다음과 같다.



<요구사항을 개념적으로 모델링하여 표현한 E-R 다이어그램>

2.1.2. 논리적 설계

논리적 설계 단계에서는 관계 데이터 모델을 이용하여 개념적 설계 단계의 결과물인 E-R 다이어그램을 관계 데이터 모델의 릴레이션 스키마인 테이블 스키마로 변환한다. 관계 데이터 모델에서는 개체와 관계를 구분하지 않고 모두 릴레이션으로 표현한다. 또한 다이어그램을 릴레이션 스키마로 변환할 때에 적용해야할 규칙 다섯 가지가 있는데, 다음과 같다.

가. 모든 개체는 릴레이션으로 변환한다.
나. 다대다 관계는 릴레이션으로 변환한다.
다-1. 일반적인 일대다 관계는 외래키로 표현한다.
다-2. 약한 개체가 참여하는 일대다 관계는 외래키를 포함해서 기본키로 지정한다.
라-1. 일반적인 일대일 관계는외래키를 서로 주고받는다.
라-2. 일대일 관계에 필수적으로 참여하는 개체의 릴레이션만 외래키를 받는다.
라-3. 모든 개체가 일대일 관계에 필수적으로 참여하면 릴레이션 하나로 합친다.
마. 다중 값 속성은 릴레이션으로 변환한다.

우리가 표현할 세 가지 관계인 외출신청, 식사신청, 반입신청은 모두 한 생도가 여러 관계를 맺을 수 있는 일대다 관계이므로 규칙 다에 초점을 중심을 두어야 한다. 가장 먼저 규칙 가를 적용해야 한다. 규칙 가에 따르면 모든 개체를 릴레이션으로

변환해야 한다. 우리의 다이어그램에는 생도, 외출, 식사, 물품 총 4개의 개체가 있으며 변환 결과는 다음과 같다.

생도 릴레이션	교번	이름	나이	성별	중대	학년	학과
외출 릴레이션	식별번호	외출종류	귀향/귀영	일자	시각	배차	
식사 릴레이션	식별번호	식사구분	날짜	메뉴			
물품 릴레이션	관리번호	물품명	반입일	반출예정일			

<규칙 가 적용한 결과>

외출, 식사, 물품 개체는 생도 개체와 일대다 관계를 맺고 있다. 따라서 규칙 다-1을 적용해야 할 것이다. 그 결과는 다음과 같다.

생도 릴레이션	교번	이름	나이	성별	중대	학년	학과			
외출 릴레이션	식별번호	외출종류	귀향/귀영	일자	시각	배차	외출자	행선지	최종결재자	확인관
식사 릴레이션	식별번호	식사구분	날짜	메뉴	신청자	취식/결식	사유			
물품 릴레이션	관리번호	물품명	반입일	반출예정일	관리자	사유	인가자			

<규칙 다-1 적용한 결과>

앞선 그림과 달리 추가된 색이 다른 부분이 일대다 관계를 고려하여 추가된 관계 속성 부분이다. 그중 빨간 색 박스 안에 들어 있는 것이 외래키로 생도 개체의 기본키인 교번 속성과 연결되어 있다. 다른 규칙 나, 라, 마는 다이어그램에 해당되지 않으므로 적용할 필요가 없다.

2.1.3. 구현

구현에 앞서서 물리적 단계가 고려되어야 하는데, 물리적 설계는 하드웨어나 운영체제의 특성을 고려하여 필요한 인덱스의 구조나 내부 저장 구조 등에 대한 물리적인 구조를 설계하는 단계[2]이다. 우리는 고려할 필요 없는 단계로, 바로 구현하면 된다. 오른쪽은 생도 릴레이션을 구현하기 위한 SQL문이다.

```
CREATE TABLE "생도" (
    "교번"    INTEGER NOT NULL,
    "이름"    TEXT,
    "나이"    INTEGER,
    "성별"    TEXT,
    "중대"    INTEGER,
    "학년"    INTEGER,
    "학과"    TEXT,
    PRIMARY KEY("교번")
);
```

<생도 릴레이션 구현을 위한 SQL문>

2) 한빛아카데미, 「데이터베이스 개론 2판」, 김현희, 340쪽.

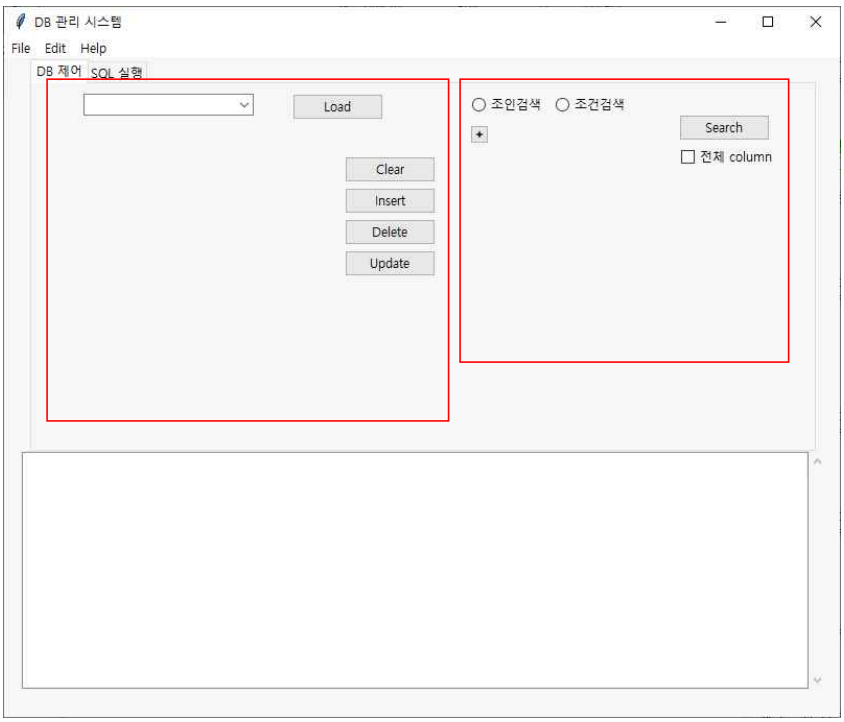
2.2. 프로그램 설계

2.2.1. 설계 철학

철학이라고 해서 거창한 것은 아니다. 프로그램 설계를 할 때에 고려해야 할 것들, 그 목적들을 정리해 둔 것뿐이다.

먼저 프로젝트 목적 파트에서 말했듯이 사용자에 친화된 인터페이스, GUI를 설계한다면 사용자의 모든 상황과 행동 가능성을 다 고려해서 프로그램이 오작동을 일으키지 않도록 예외에 대한 처리를 해주는 것이 바람직하다. 또한 인터페이스의 형태도 특정 데이터베이스에 맞추는 것보다 범용적인 형태로 개발하고자 한다. 물론 대상으로 한 데이터베이스가 이미 존재하다 보니 대체적으로 그것에 맞춰서 개발을 진행하게 될 것이다. 다만, 프로그램이 다른 데이터베이스를 다루게 되더라도 최소한의 오작동으로 동작을 수행할 수 있도록 만드는 것이 목적이다.

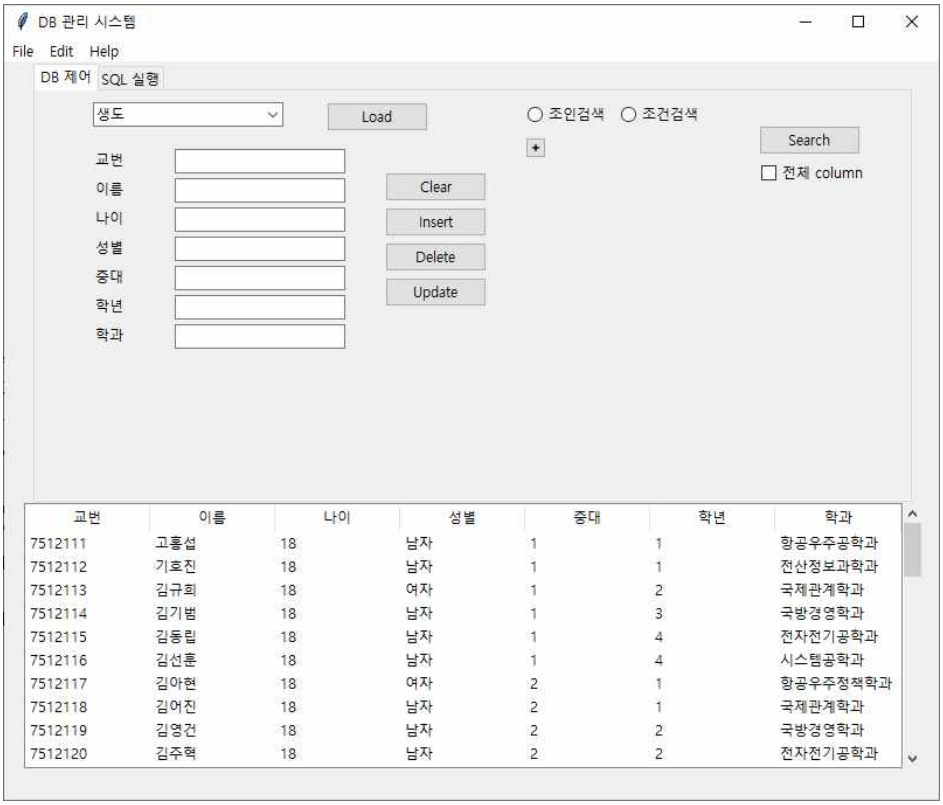
2.2.2. 출력 화면



<실행 초기 화면>

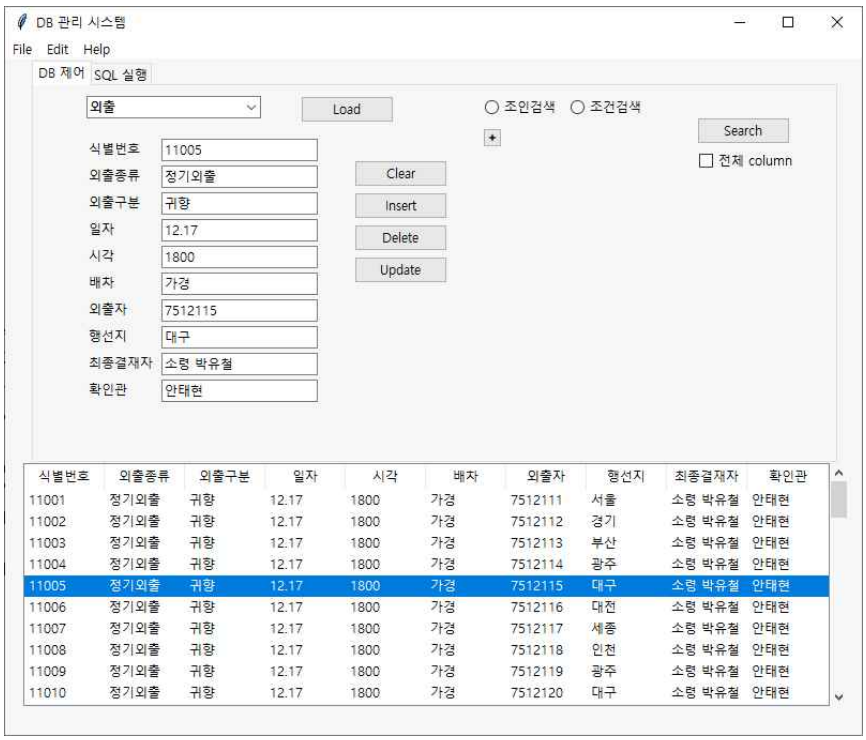
실행 초기 화면이다. 가장 먼저 위의 File 메뉴에서 Open을 이용해 데이터베이스 파일을 불러와야 한다. 불러오고나면 첫 번째 테이블이 불러와지며 왼쪽 빨간 박스의 콤보박스에서는 테이블을 선택할 수 있게 되고 그 아래로 컬럼마다 해당하는 라벨과 엔트리가 나타난다. 그리고 아래에 있는 트리뷰에는 불러와진 테이블의 내용이 들어간다.

다음은 파일을 불러오고 난 직후의 화면이다.

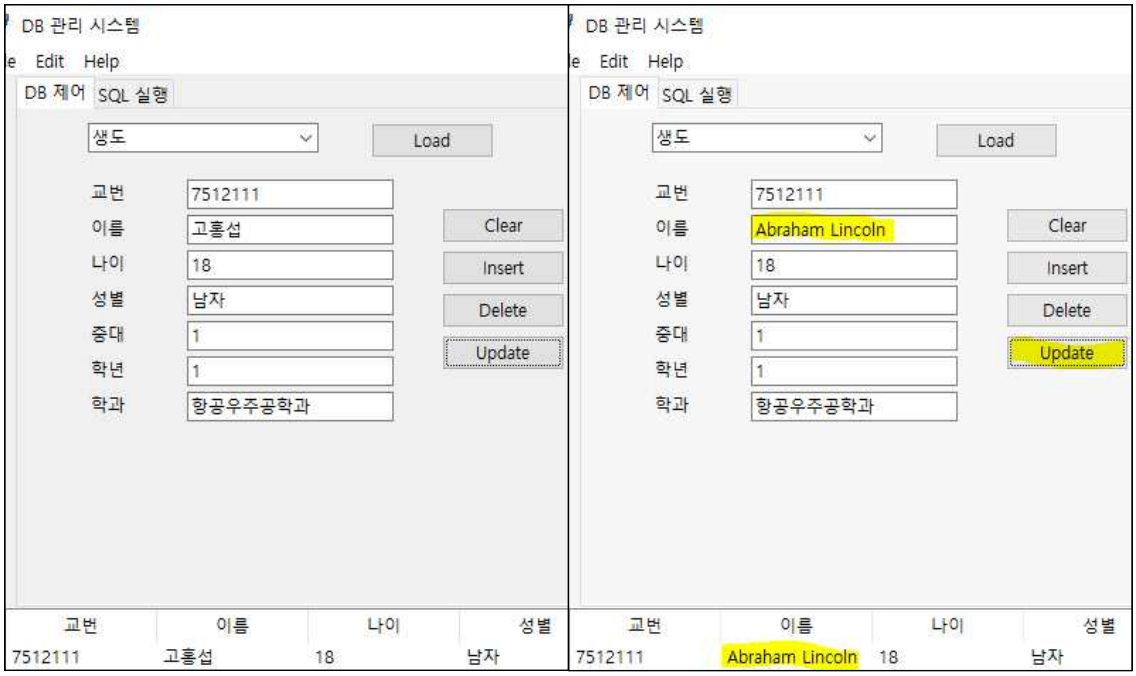


<생도대.db를 불러오면서 생도 테이블이 나오는 모습>

밑의 트리뷰에서 행을 선택하면 행 해당하는 내용들이 위의 엔트리 안에 들어가게 된다. 다음이 그 모습이다.

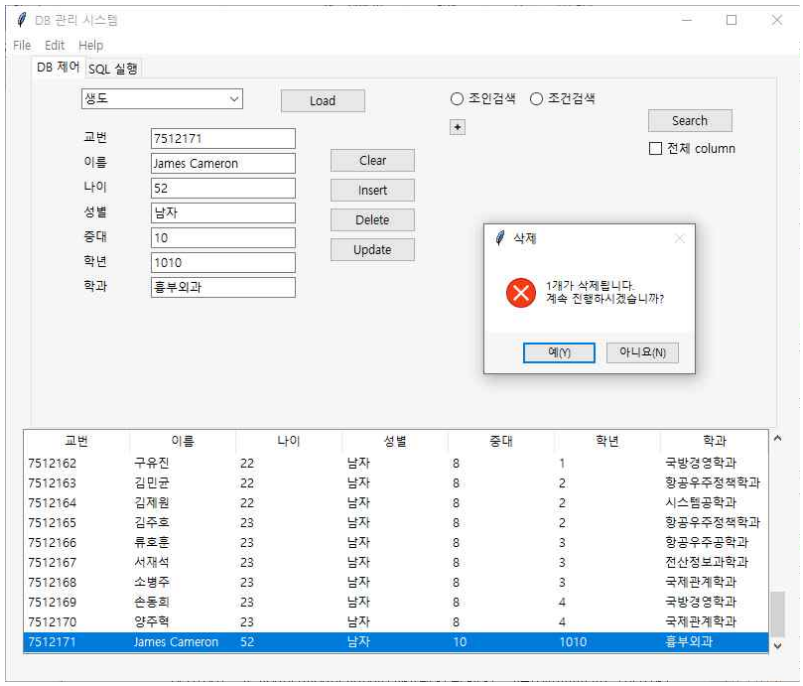


<트리뷰 클릭에 따른 이벤트 처리>



<Update 동작 수행 모습>

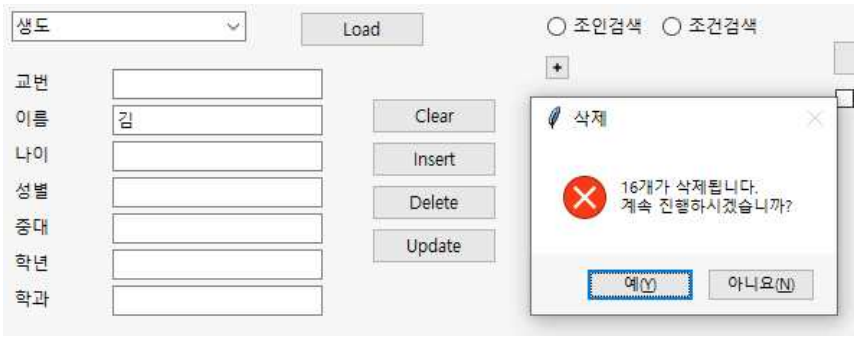
이름 항목에 대한 Update 수행 모습이다. 모든 행에 대하여도 얼마든지 동작을 수행할 수 있다. 참고로 Clear 버튼은 저기 엔트리에 있는 내용물들을 지우는 동작을 수행한다.



<Delete 동작 수행 모습>

삭제 동작인 Delete의 경우 신경을 썼다. 그림에 나와 있는 것처럼 트리뷰의 행을

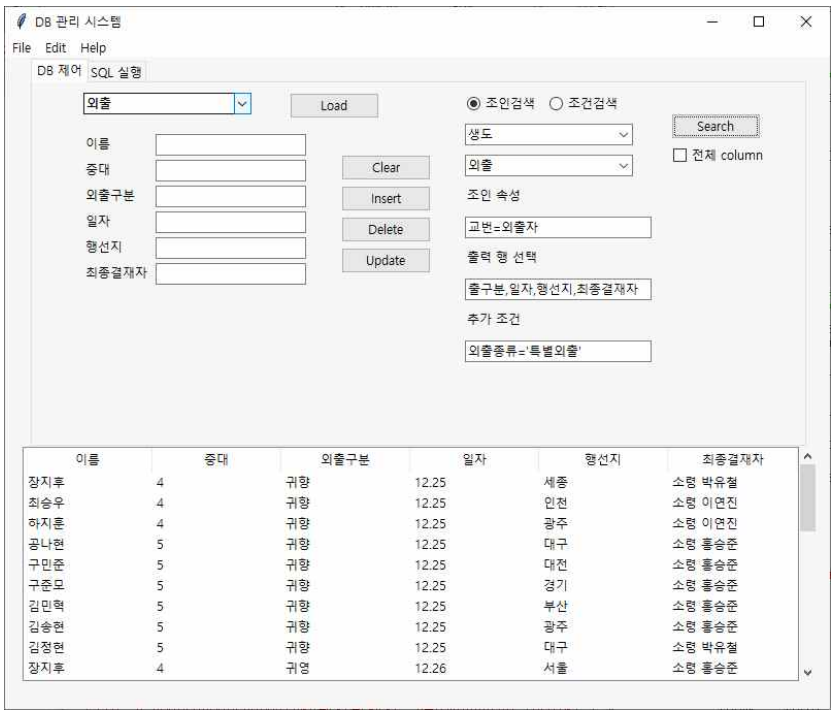
선택해서 삭제를 수행하면 엔트리가 꽉 차있는 것이 맞지만 만약 하나의 특수한 행만 삭제하는 것이 아니라, 예를 들어, 이름에 '김'이 들어가는 행을 모두 지우고 싶다면 Clear로 엔트리를 다 비운다음 이름 엔트리에 '김'을 넣고 삭제 버튼을 누르면 다음과 같이 동작한다.



<여러 항목을 동시에 Delete 하는 모습>

이름에 '김'이 들어가는 16개 항목이 모두 삭제될 거라고 경고를 하며 '예'를 선택하면 삭제가 진행된다.

다음으로 볼 것은 검색이다.

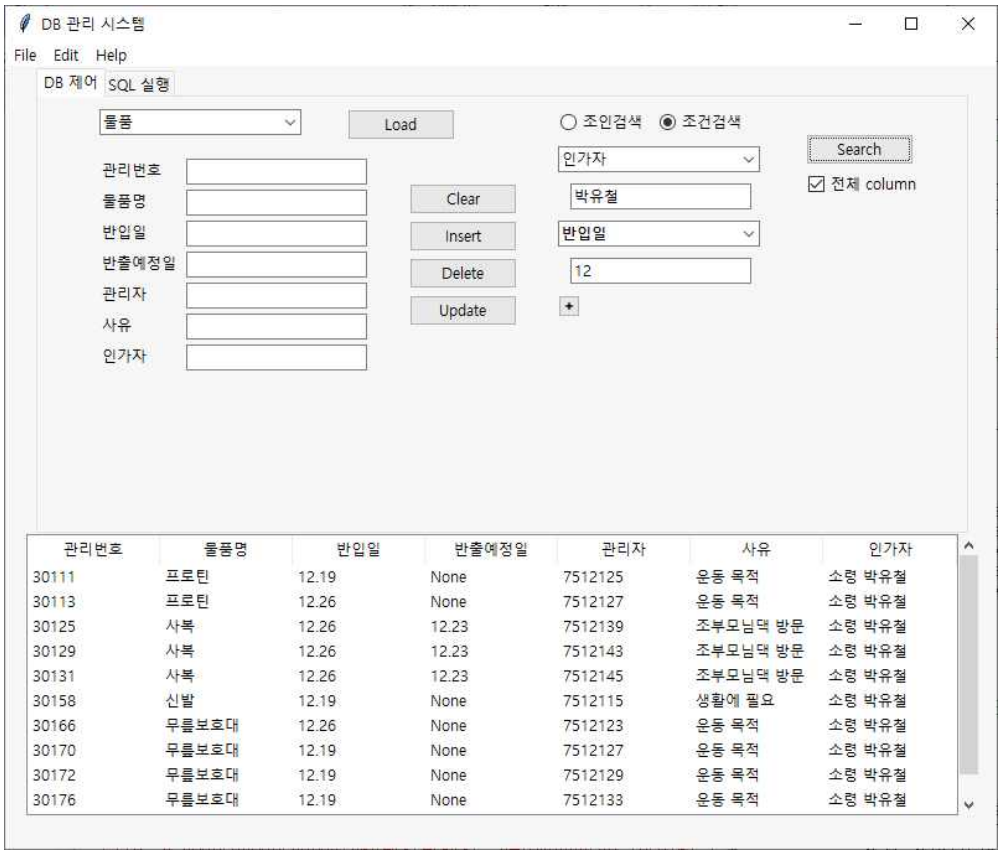


<조인검색을 수행하는 모습>

라디오버튼으로 조인검색, 조건검색 중 하나를 선택할 수 있다. 위 사진은 그 중 조인검색의 경우이다. 조인검색을 선택하면 비어있던 부분에 위젯들이 나타나는데, 상단에 위치한 두 개의 콤보박스는 조인을 시킬 두 테이블을 고르도록 유도한 것이

다. 그 아래는 라벨들에 적혀있듯 조인 속성에 대해 묻는 것으로 테이블에 다양한 외래키가 있을 것을 염두하여 조인할 속성에 대해 사용자에게 직접 묻는 것이다. 그 다음 출력 행 선택인데, 내가 보고 싶은 행만 고를 수 있도록 입력을 받는다. 단, 옆에 Search 버튼 아래에 있는 체크 박스가 전체 행을 볼 수 있게 해주며 이것에 더 우선권이 있다. 만약 이를 체크하면 출력 행 입력 부분에 뭘 입력하더라도 모든 행을 보여준다. 그 다음은 추가 조건으로 조인에서 그치지 않고 조건을 추가로 주고 싶다면 SQL문의 WHERE 절에 나오는 것처럼 입력을 넣어주면 된다.

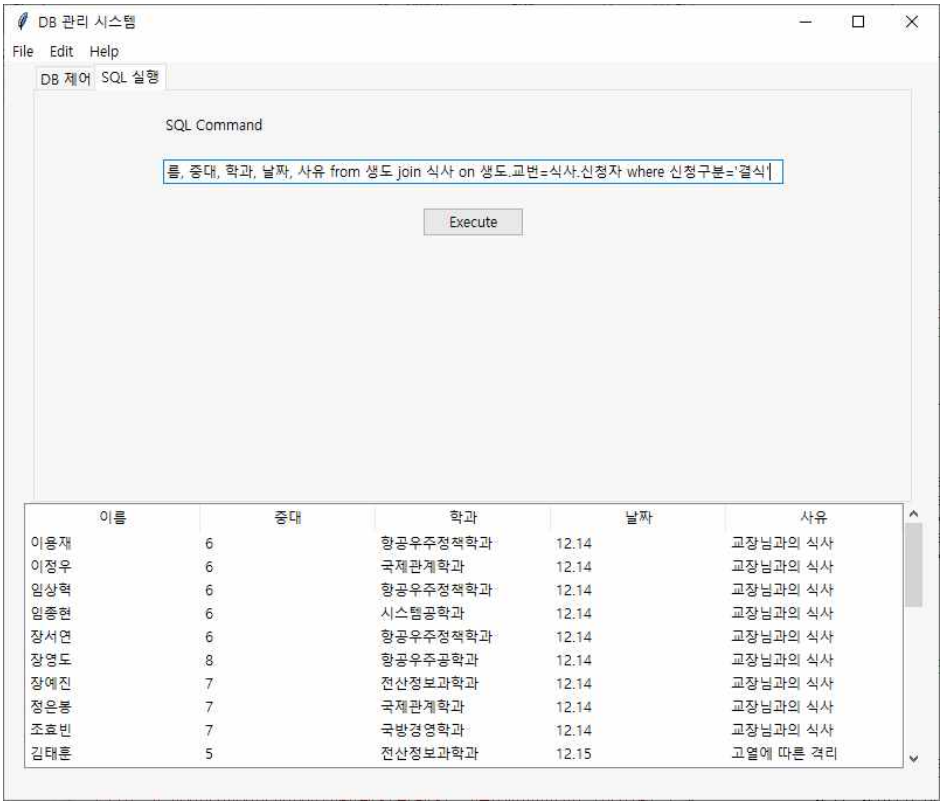
다음은 조건검색이다.



<조건검색을 수행하는 모습>

라디오 버튼에서 조건검색을 누르면 +표시가 된 작은 버튼이 생긴다. 이를 누르면 콤보박스와 엔트리가 하나씩 생기고 그 밑에 다시 +버튼이 누를 때마다 한 쌍씩 밑으로 반복해서 생기는 형태이다. 이들의 역할은 콤보 박스로 열을 고르고, 엔트리에 검색하고 싶은 조건을 받는 것이다. 콤보박스에 생기는 열은 현재 보고 있는 테이블의 열들을 따라간다. 또한 엔트리에 입력한 값을 일부로 포함한 것들도 다 검색하기 때문에 위의 사진은 인가자가 '박유철'이고 반입일에 '12'가 들어간 모든 항목을 검색한 것이다. 이 검색도 체크박스로 출력을 조절할 수 있다. 만약 체크박스를 체크하지 않으면 조건 검색에 쓰인 열만 출력되게 된다.

마지막으로 SQL문을 직접 입력 받는 경우이다.



<SQL문을 직접 입력받는 모습>

SQL Command라는 이름을 가진 엔트리에 SQL문을 입력하고 Execute 버튼을 눌러주면 입력한 대로 동작을 수행하여 결과를 트리뷰에 출력해준다. 사진은 조인 검색을 이용한 검색의 모습이다.

2.3.1. 코드에 대한 간략한 설명

프로그램 크기가 커지다 보니 코드만 470 line가량 된다. 이 정도 크기만 되도 변수들과 함수의 호출 관계를 단번에 파악하기란 어렵다. 게다가 대부분의 동작을 함수의 호출과 반환의 연속으로 이뤄진 GUI 프로그램은 더더욱 그렇다. 코드의 전문은 부록으로 첨부하고, 간략하게 설명해 보고자한다.

코드는 크게 3개의 class와 함수부, 메인부로 나눌 수 있다. 가장 먼저 메인부는 프로그램 실행 자체를 위해 있다. 파이썬 Tkinter는 GUI를 작동시키위해 메인부에서 계속해서 루프를 돌리게 된다. 이 메인 부 안에 초기 화면에 나오는 위젯들을 넣어주고 각 위젯들에게 발생한 이벤트를 처리할 함수들을 부여해준다. 이 부분에는 단순히 메인 창을 만들고, 위젯을 위치시키는 정도라 파악이 어렵지도 않고 중요한 분도 아니다.

그 다음으로 함수부는 위젯들과 연결되어 이벤트가 발생하면 작동하는 함수들을 모아둔 것이다. 여기에 있는 함수들은 class의 객체들을 통해 실제 동작을 수행하는 메소드들을 불러가며 사용자가 요구하는 동작들을 수행하게 된다.

여기서 의문을 가질지 모르겠다. 객체의 메소드라는 것은 결국 그것도 함수인데 왜 함수부에 있는 함수들을 이용하지 않고 따로 class를 만드느냐고 말이다. 잘 알고 있겠으나 만약 함수들로만 모든 동작을 수행하려 한다면 전역변수가 남발할 수 밖에 없다.

GUI 프로그램은 그 특성상 수많은 위젯들은 각각의 함수를 갖고 그 함수 안에서 다양한 지역 변수들을 만든다. 안타깝게도 위젯들의 동작이 복잡하면 복잡할수록 이 지역 변수들은 함수 바깥으로 나가 다른 함수에게 전달되어야 하는 경우가 많다. 단순하게 리턴과 패러미터로 해결이 되면 좋을텐데, 위젯들은 인자를 받지 않는다.

여기서 전역변수의 필요가 생겨버린다. 결국 복잡한 동작을 요구하게 될수록 전역변수가 남발하게 되고 변수가 겹치지 않게 변수명 짓기에 아이디어를 쏟아부을 것이다. 어쨌든 전역 변수가 많아지는 건 프로그램에 좋지 않다. 안그래도 읽기 힘든 코드가 더 읽기 힘들어지고 디버깅은 물론 후에 유지, 보수도 힘들어진다.

이를 해결할 방법이 바로 class이다. class는 그 안에서 self.를 붙여 만든 변수들은 class내에서 계속해서 살아간다. 그 객체가 폐기 되지 않는 한 말이다. class내에서 만큼은 전역변수처럼 행동하는 것이다. 따라서 특정한 값들을 여러 함수에서 공통적으로 가져다 쓴다면 그 함수들을 묶어 class로 만들어주는 것이 효율적이라는 것이다.

그러한 결과로 생긴 것이 database, widgets, Searching 클래스 들이다. 비슷한 맥락의 함수들 끼리 묶어서 복잡한 함수들이 아무리 많아도 최소한의 전역변수들 만으로 최대한의 효율을 만들어 냈다.

3. 결론

한 학기에 걸쳐서 이루어진 프로젝트가 하나의 완성된 데이터베이스 관리 프로그램을 낳고 끝났다. 학기가 시작할 때 소개받은 프로젝트의 모습은 내가 저걸 만들 수 있을까 하는 두려움 반, 어떻게 하면 저걸 만들 수 있을까 하는 기대 반이었다. 다른 수업의 과제들과 생활에서 요구하는 많은 일과들로 인해 뺏긴 시간은 프로젝트에게 아쉬움을 남겼다. 조금만 더 시간이 있었으면 더 고민해서 나은 결과물을 만들 수 있지 않을까 하는 아쉬움이다.

또 다른 아쉬움도 있다. 2학기에 수업시간은 유독 부족했다. 더 긴 시간동안 수업을 들을 수 있었다면 데이터베이스도 시간 들여서 설계 원칙에 맞게 만들 수 있었을 텐데 하는 아쉬움이 있다. 그럼에도 2학기라는 짧지 않은 시간동안 배운 모든 것을 지금의 이 프로젝트에 다 쏟아내어 잘 섞고 혼합하여 멋있게 잘 만들어 낸거 같다. 흔히들 치르는 필기고사 식의 시험과는 달리 결과물이 또렷하게 남고 하나의 소중한 경험이 되어 미래에도 계속해서 내게 남아 있을 거라는 확신이 든다.

참고문헌

George El, ‘Create a Tkinter Gui With Sqlite Backend’, https://www.python4networkengineers.com/posts/python-intermediate/create_a_tkinter_gui_with_sqlite_backend/.

한빛아카데미, 「데이터베이스 개론 2판」, 김현희.

부록

코드

```
from tkinter import *
from tkinter import messagebox
from tkinter import filedialog
from tkinter.ttk import *
import sqlite3
##########################################################################
class widgets:
    def __init__(self, database):
        self.db = database
        self.entry = {}
        self.label = {}
        self.init_table = database.getTableName()[0]
        self.init_column = database.getColumnName(self.init_table)
        table_cbox.set(self.init_table)
        table_view.destroy()
        scrollbar.destroy()
        self.creat_widget(self.init_column)
        self.print_view(self.init_column)
        self.print_row(self.init_table)

    def select_tree(self, event):
        try:
            col_list = self.db.getColumnName(table_cbox.get())
            index = self.table_view.selection()[0]
            self.selected_item = self.table_view.item(index)['values']
            for i, col in enumerate(col_list):
                self.entry[col].delete(0, END)
                self.entry[col].insert(END, self.selected_item[i])
        except:
            try:
                col_list = jc.con_clist
            except:
                col_list = jc.join_clist
            index = self.table_view.selection()[0]
            self.selected_item = self.table_view.item(index)['values']
            for i, col in enumerate(col_list):
                self.entry[col].delete(0, END)
                self.entry[col].insert(END, self.selected_item[i])

    def get_value(self):
        value_list = []
        for col in self.past_cols:
            value_list.append(self.entry[col].get())
        return value_list

    def table_load(self):
        tname = table_cbox.get()
        columns = self.db.getColumnName(tname)
        self.clear_widget()
        self.print_view(columns)
        self.print_row(tname)
        self.creat_widget(columns)
```

```
    def print_view(self, cols):
        self.table_view = Treeview(frame3, columns=cols, show='headings')
        len_fit = int(750/len(cols))
        for col in cols:
            self.table_view.column(col, width=len_fit)
            self.table_view.heading(col, text=col)
        self.table_view.bind('<<TreeviewSelect>>', self.select_tree)
        self.table_view.pack(side="left", fill='y')
        self.scrollbar = Scrollbar(frame3, orient='vertical')
        self.scrollbar.config(command=self.table_view.yview)
        self.scrollbar.pack(side="right", fill="y")
        self.table_view.config(yscrollcommand=self.scrollbar.set)

    def print_row(self, tname, sql=None):
        try:
            for i in self.table_view.get_children():
                self.table_view.delete(i)
        except:
            pass
        if sql != None:
            for row in sql:
                self.table_view.insert('', 'end', values=row)
        else:
            for row in self.db.tv_content(tname):
                self.table_view.insert('', 'end', values=row)

    def creat_widget(self, cols):
        for i, col in enumerate(cols):
            lb = Label(frame1, text=col)
            lb.place(x=50, y=50+i*25)
            self.label[col] = lb
            e = Entry(frame1)
            e.place(x=120, y=50+i*25)
            self.entry[col]=e
        self.past_cols = cols

    def clear_widget(self):
        self.table_view.destroy()
        self.scrollbar.destroy()
        for col in self.past_cols:
            self.label[col].destroy()
            self.entry[col].destroy()

    def clear_entry(self):
        for col in self.past_cols:
            self.entry[col].delete(0, END)
################################################################################
class database:
    def __init__(self, dbfile):
        self.conn = sqlite3.connect(dbfile)
        self.cur = self.conn.cursor()
        self.cur.execute('SELECT NAME FROM sqlite_master WHERE TYPE="table"')
        qd = self.cur.fetchall()
        self.table_list = [tb[0] for tb in qd]

    def getTableName(self):
        return self.table_list

    def getColumnName(self, table_name):
```

```
        self.cur.execute("SELECT * FROM %s LIMIT 1;" %table_name)
        field_list = [fd[0] for fd in self.cur.description]
        return field_list

    def getPrimary(self, table_name):
        self.cur.execute("PRAGMA table_info({})".format(table_name))
        cinfo = self.cur.fetchall()
        for c in cinfo:
            if c[-1] == 1:
                return c[1]
            else:
                return None

    def getType(self, table_name):
        typeinfo = {}
        self.cur.execute("PRAGMA table_info({})".format(table_name))
        cinfo = self.cur.fetchall()
        for c in cinfo:
            if c[2] == 'INTEGER':
                typeinfo[c[1]] = 'int'
            else:
                typeinfo[c[1]] = 'str'
        return typeinfo

    def tv_content(self, table_name):        #테이블 전체 내용 리턴
        self.cur.execute("SELECT * FROM %s;" %table_name)
        rows = self.cur.fetchall()
        return rows
    def insert(self, table_name, value_list):
        blank = '('+(",".join('?'*len(value_list)))+')'
        self.cur.execute("INSERT INTO {} VALUES ".format(table_name)+blank, tuple
([value for value in value_list]))
        self.conn.commit()
    def erase(self, table_name, value_list):
        clist=self.getColumnName(table_name)
        condition=[]
        for i, value in enumerate(value_list):
            if value != '':
                condition.append(clist[i]+' LIKE "%'+value+'%"')
        constr=" and ".join(condition)
        self.cur.execute("SELECT COUNT(*) AS num FROM {} WHERE {}".format(table_n
ame, constr))
        num = self.cur.fetchone()[0]
        msgbox=messagebox.askquestion('삭제','{}개가 삭제됩니다.\n계속 진행하시겠습
니까?'.format(num),icon='error')
        if msgbox == 'yes':
            self.cur.execute("DELETE FROM {0} WHERE {1}".format(table_name, const
r))
            self.conn.commit()
        else:
            pass
    def update(self, table_name, value_list):
        clist=self.getColumnName(table_name)
        setstr = ''
        for c in clist[1:]:
            setstr += '{}=?,'.format(c)
        self.cur.execute("UPDATE {} SET ".format(table_name)+setstr[:-1]+" WHERE
{}=?".format(self.getPrimary(table_name)),  tuple([value for value in value_list[
1:]]+[value_list[0]]))
```

```
        wg.print_row(table_name)
        self.conn.commit()
    def execute(self, command):
        try:
            self.cur.execute(command)
        except:
            messagebox.showerror("Command Error", "잘못된 SQL문을 입력하였습니다.")
            return False
        rows = self.cur.fetchall()
        return rows
    def __del__(self):
        self.conn.close()
#################################################################
class Searching:
    def __init__(self):
        self.search_num = 0
        self.join_num = 0

    def changeColumn(self, n):
        try:
            tname = table_cbox.get()
        except:
            return
        clist = db.getColumnName(tname)
        exec("self.search_cbox{}['values'] = clist".format(n))

    def con_search(self):
        self.con_clist = []
        self.con_conlist = []
        self.con_column = ''
        self.con_condition = ''
        for i in range(1, self.search_num+1):
            exec("self.con_clist.append(self.search_cbox{}.get())".format(i))
            exec("self.con_column += self.search_cbox{}.get()+','".format(i))
            exec("""self.con_condition += self.search_cbox{0}.get()+' like '+'"%'
+self.search_entry{0}.get()+'%"'+' and '""".format(i))
        if chkvar.get():
            cmd="SELECT * FROM "+table_cbox.get()+" WHERE "+self.con_condition[:-
5]
        else:
            cmd="SELECT "+self.con_column[:-1]+" FROM "+table_cbox.get()+" WHERE
"+self.con_condition[:-5]
        return cmd

    def join_search(self):
        t1 = self.join1.get()
        t2 = self.join2.get()
        outcol = self.colVar.get()
        attr = self.joinVar.get()
        addcon = self.addVar.get()
        if chkvar.get():
            cmd = "SELECT * FROM "+t1+" INNER JOIN "+t2+" ON "+attr
        else:
            cmd = "SELECT "+outcol+" FROM "+t1+" INNER JOIN "+t2+" ON "+attr
        if addcon != '':
            cmd += " WHERE "+addcon
        self.join_clist = outcol.replace(',',' ').split()
        return cmd
```

```
    def next_condition(self):
        add_btn.place_forget()
        if self.search_num < 5:
            self.search_num += 1
            exec("""values=db.getColumnName(table_cbox.get())\n
self.search_cbox{0}=Combobox(frame1, values=values)\n
self.search_cbox{0}.place(x=420, y=40+60*({0}-1))\n
self.search_var{0}=StringVar()\n
self.search_entry{0}=Entry(frame1, textvariable=self.search_var{0}, width=20)\n
self.search_entry{0}.place(x=430, y=10+60*{0})""".format(self.search_num))
            add_btn.place(x=420, y=40+60*self.search_num)
        if self.search_num == 5:
            add_btn.place_forget()

    def join_to_con(self):
        try:
            self.join1.destroy()
            self.join2.destroy()
            self.joinLabel.destroy()
            self.joinEntry.destroy()
            self.colLabel.destroy()
            self.colEntry.destroy()
            self.addLabel.destroy()
            self.addEntry.destroy()
            add_btn.place(x=420, y=40)
        except:
            pass

    def con_to_join(self):
        for i in range(1,self.search_num+1):
            exec("""self.search_cbox{0}.destroy()\n
self.search_entry{0}.destroy()""".format(i))
        self.search_num = 0

    def join_set(self):
        try:
            self.con_to_join()
        except:
            pass
        add_btn.place_forget()
        values=db.getTableName()
        self.join1=Combobox(frame1, values=values)
        self.join1.place(x=420, y=40)
        self.join2=Combobox(frame1, values=values)
        self.join2.place(x=420, y=70)
        self.joinLabel=Label(frame1, text='조인 속성')
        self.joinLabel.place(x=420, y=100)
        self.joinVar=StringVar()
        self.joinEntry=Entry(frame1, textvariable=self.joinVar, width=25)
        self.joinEntry.place(x=420, y=130)
        self.colLabel=Label(frame1, text='출력 행 선택')
        self.colLabel.place(x=420, y=160)
        self.colVar=StringVar()
        self.colEntry=Entry(frame1, textvariable=self.colVar, width=25)
        self.colEntry.place(x=420, y=190)
        self.addLabel=Label(frame1, text='추가 조건')
        self.addLabel.place(x=420, y=220)
        self.addVar=StringVar()
        self.addEntry=Entry(frame1, textvariable=self.addVar, width=25)
```

```
        self.addEntry.place(x=420, y=250)

def clear_db():
    wg.clear_entry()

def insert_db():
    tname = table_cbox.get()
    vlist = wg.get_value()
    db.insert(tname, vlist)
    wg.clear_entry()
    wg.print_row(tname)

def erase_db():
    tname = table_cbox.get()
    vlist = wg.get_value()
    db.erase(tname, vlist)
    wg.clear_entry()
    wg.print_row(tname)

def update_db():
    tname = table_cbox.get()
    vlist = wg.get_value()
    db.update(tname, vlist)

def search_db():
    if Searchvariable.get() == 'JOIN':
        cmdline = jc.join_search()
        rows = db.execute(cmdline)
        wg.clear_widget()
        if chkvar.get():
            wg.print_view(db.getColumnName(jc.join1.get())+db.getColumnName(jc.jo
in2.get()))
            wg.creat_widget(db.getColumnName(jc.join1.get())+db.getColumnName(jc.
join2.get()))
        else:
            wg.print_view(jc.join_clist)
            wg.creat_widget(jc.join_clist)
        wg.print_row(cmdline, rows)
    else:
        cmdline = jc.con_search()
        rows = db.execute(cmdline)
        wg.clear_widget()
        if chkvar.get():
            wg.print_view(db.getColumnName(table_cbox.get()))
            wg.creat_widget(db.getColumnName(table_cbox.get()))
        else:
            wg.print_view(jc.con_clist)
            wg.creat_widget(jc.con_clist)
        wg.print_row(cmdline, rows)

def execute_db():
    cmd = cmdline.get()
    rows = db.execute(cmd)
    ex_cmd = cmd.replace(',', ' ')
    ex_cmdL = ex_cmd.lower().split()
    tcount = 0
    tlist = []
    cols = []
    for ss in ex_cmdL:
```

```
        if ss in db.getTableName():
            tlist.append(ss)
            tcount += 1
    if ex_cmdL[1] == '*':
        if tcount > 1:
            for i in range(len(tlist)):
                cols += db.getColumnName(tlist[i])
        else:
            cols = db.getColumnName(tlist[0])
    else:
        end = ex_cmdL.index('from')
        for col in ex_cmdL[1:end]:
            cols += [col]
    if rows == False:
        return
    wg.clear_widget()
    wg.print_view(cols)
    wg.print_row(cmd, rows)

def Load():
    filename = filedialog.askopenfilename(initialdir="/", title="Select file", fi
letypes=(("db files", "*.db"), ("all files", "*.*")))
    global db
    db = database(filename)
    global wg
    wg = widgets(db)

def Save():
    filename = filedialog.asksaveasfilename(initialdir="/", title="Select file",
filetypes=(("db files", "*.db"), ("all files", "*.*")))

def domenu():
    print("OK")

def table_load():
    wg.table_load()

###페이지1
def changeTable():
    tlist = db.getTableName()
    table_cbox['values'] = tlist

root = Tk()
root.title("DB 관리 시스템")
root.geometry("800x630+100+100")
plus_image=PhotoImage(file="plus_icon8.png")
jc = Searching()
###메뉴
menubar = Menu(root)                                    # 윈도우에 메뉴바 추가
filemenu = Menu(menubar, tearoff=0)                     # 상위 메뉴 탭 항목 추가
menubar.add_cascade(label="File", menu=filemenu)        # 상위 메뉴 탭 설정
filemenu.add_command(label="New", command=domenu)       # 항목 추가
filemenu.add_command(label="Open", command=Load)
filemenu.add_command(label="Save", command=Save)
filemenu.add_command(label="Save as...", command=domenu)
filemenu.add_separator()                                # 분리선 추가
filemenu.add_command(label="Exit", command=root.quit)
editmenu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="Edit", menu=editmenu)
```

```
editmenu.add_command(label="Copy", command=domenu)
editmenu.add_command(label="Paste", command=domenu)
editmenu.add_separator()
editmenu.add_command(label="Delete", command=domenu)
helpmenu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="Help", menu=helpmenu)
helpmenu.add_command(label="About...", command=domenu)
root.config(menu=menubar)                          # 생성된 객체를 위에서 생성된 메
뉴바에 연결
###페이지탭
frame0=Frame(root)
frame0.grid(row=0, column=0, padx=25)
notebook=Notebook(frame0, width=750, height=350)
notebook.pack()
frame1=Frame(root)
notebook.add(frame1, text="DB 제어")
table_cbox=Combobox(frame1, height=15, values=[], postcommand=changeTable)
table_cbox.place(x=50, y=10)
table_btn=Button(frame1, text='Load', command=table_load).place(x=250, y=10)
clear_btn=Button(frame1, text='Clear', command=clear_db).place(x=300, y=70)
insert_btn=Button(frame1, text='Insert', command=insert_db).place(x=300, y=100)
erase_btn=Button(frame1, text='Delete', command=erase_db).place(x=300, y=130)
update_btn=Button(frame1, text='Update', command=update_db).place(x=300, y=160)
search_btn=Button(frame1, text='Search', command=search_db).place(x=620, y=30)
chkvar = BooleanVar()
chkvar.set(False)
a l l _ c h b t n = C h e c k b u t t o n ( f r a m e 1 ,   t e x t = ' 전 체
 column', var=chkvar, onvalue=True, offvalue=False).place(x=620, y=60)
Searchvariable=StringVar()
j o i n _ r b = R a d i o b u t t o n ( f r a m e 1 ,   t e x t = " 조 인 검 색
", value='JOIN', variable=Searchvariable, command=jc.join_set)
join_rb.place(x=420, y=10)
c o n _ r b = R a d i o b u t t o n ( f r a m e 1 ,   t e x t = ' 조 건 검 색
', value='WHERE', variable=Searchvariable, command=jc.join_to_con)
con_rb.place(x=500, y=10)
add_btn=Button(frame1, image=plus_image, width=20, command=jc.next_condition)
add_btn.place(x=420, y=40)
frame2=Frame(root)
notebook.add(frame2, text="SQL 실행")
sqllb=Label(frame2, text="SQL Command")
sqllb.grid(row=0, column=0, sticky=W, padx=110, pady=20)
cmdline=StringVar()
sqlcmd=Entry(frame2, textvariable=cmdline, width=75)
sqlcmd.grid(row=1, column=0, padx=110)
sqlbtn=Button(frame2, text='Execute', command=execute_db)
sqlbtn.grid(row=2, column=0, padx=110, pady=20)
###트리뷰
col=[' ']
frame3=Frame(root)
frame3.grid(row=1, column=0)
table_view = Treeview(frame3, columns=col, show='headings')
table_view.column(col[0], width=750)
table_view.heading(col[0], text=col)
table_view.pack(side="left", fill="y")
scrollbar = Scrollbar(frame3, orient='vertical')
scrollbar.configure(command=table_view.yview)
scrollbar.pack(side="right", fill="y")
table_view.config(yscrollcommand=scrollbar.set)
```