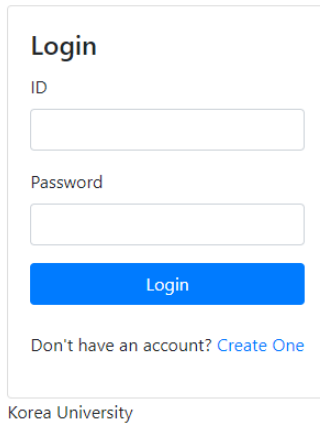


COSE371 데이터베이스

2020320044 컴퓨터학과 백민규, 2020320078 컴퓨터학과 한지상

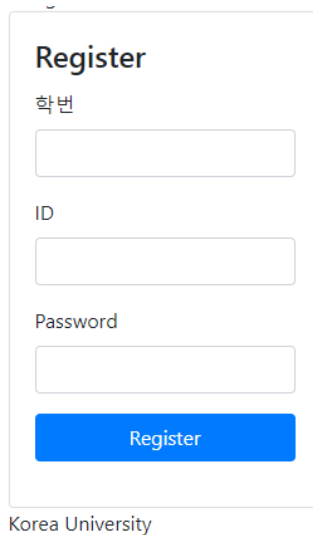
1. DB application에 대한 설명



The login form is titled "Login" and is part of the Korea University system. It contains two input fields: "ID" and "Password". Below these fields is a blue "Login" button. At the bottom of the form, there is a link that says "Don't have an account? [Create One](#)". The form is enclosed in a light gray border.

<http://127.0.0.1:5000/login> 에서 ID와 Password를 통해 로그인할 수 있다.

만약 계정이 없다면 Create One을 눌러 Register 페이지로 이동할 수 있다.



The register form is titled "Register" and is part of the Korea University system. It contains three input fields: "학번" (Student ID Number), "ID", and "Password". Below these fields is a blue "Register" button. The form is enclosed in a light gray border.

Register 페이지에서 학번과 ID, Password를 입력하여 회원가입을 하면 다시 로그인 페이지로 돌아와 로그인을 진행할 수 있다. 존재하지 않는 학번이거나, 이미 가입된 ID면 오류 메시지가 나온다.

Hello. onground

로그인 시 메인 페이지. 로그인한 ID를 확인할 수 있다. (ID: onground / PW: 12345678)

학수번호	분반	이수구분	개설학과	교과목명	담당교수	학점(시간)	강의시간	선수과목	학점
COSE3625		major_elective	컴퓨터학과	기계학습	육동석	3(3)	tuesday 7교시 우정정보관 101	COSE101	A
COSE3712		major_required	컴퓨터학과	데이터베이스이	혁기	3(3)	tuesday 8교시 우정정보관 101 thursday 7교시 우정정보관 101	COSE102A+	COSE213

MyCourses를 누르면 내가 이수한 과목의 정보를 확인할 수 있다.

2021
fall

단과대학
정보대학

공/교양
교양

학점
전체--

교시
전체--

담당교수

조회

View All Courses를 누르면 DB에 저장되어 있는 과목의 정보를 확인할 수 있다. 또한, 년도, 단과대 학(교양의 경우 단과대학 구분 없이 '교양'), 학점, 요일, 교시, 담당교수, 학수번호, 분반, 교과목명 을 기준으로 조회가 가능하다. 아무것도 입력하지 않은 상태로 조회를 누르면 모든 과목이 출력 되고, 각각의 기준을 적용하면 더욱 정밀한 조회가 가능하다. SQL like를 사용하여 각각에 대해 일 부만 작성하여 검색하여도 조회가 가능하다.

학번	이름	GPA	이메일	단과대학	전공	제2전공	지도교수	상태(재학/휴학)	전화번호	주소	우편번호	생년월일
2020320078	한지상	4.25	jlsang77747@gmail.com	정보대학	컴퓨터학과	None	김승룡	present	01054968096	서울특별시 도봉구 방학동	01337	2001-02-14

탈퇴하시겠습니까? 예

My Page를 누르면 나의 정보를 확인할 수 있다. 학번, 이름, GPA, 단과대학, 전공, 지도교수, 상태, 전화번호, 주소, 우편번호, 생년월일의 정보를 확인할 수 있다.

그리고 아래 칸에 학번을 입력하고 예 버튼을 누르면 수강과목 및 정보확인 사이트에서 탈퇴가 가능하다.

비밀번호변경

아이디

새로운비밀번호

새로운비밀번호확인

확인

Change Password를 누르면 비밀번호를 변경할 수 있는 페이지가 나온다. 사이트에 로그인할 때 사용한 ID와 새로운 비밀번호를 누르면 비밀번호 변경이 가능하다. 만약 현재 접속 아이디와 다르거나 새로운비밀번호확인이 실패할 경우 비밀번호 변경이 불가능하다.

2021

단과대학전공/교양

정보대학

학점

요일

전체

교시

전체

담당교수

조희

학수번호

분반

교과목명

과목 추가

개설년도

2021

개설학기

fall

학수번호

COSE214

분반

01

교과목명

알고리즘

이수구분

전공필수

개설학과

컴퓨터학과

담당교수

2020320005

학점

3

시간

3

요일

화

교시

5

선수과목

COSE101

건물

우정정보관

호수

101

과목 추가

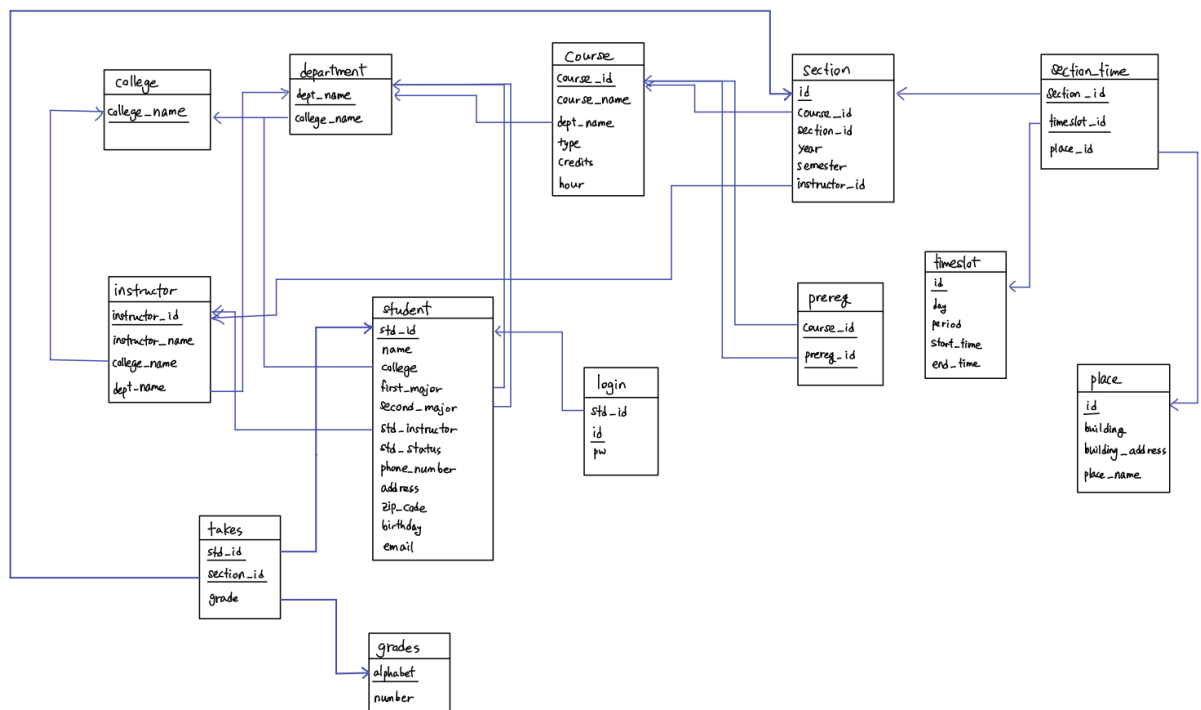
과목 삭제

개설년도		개설학기	fall	학수번호		분반		과목 삭제
학수번호	분반	이수구분	개설학과	교과목명	담당교수	학점(시간)	강의시간	선수과목
COSE213	01	major_required	컴퓨터학과	자료구조	정원기	3(3)	tuesday 5교시 thursday 5교시	COSE101 COSE102
COSE213	02	major_required	컴퓨터학과	자료구조	김선희	3(3)		COSE101 COSE102
COSE362	01	major_elective	컴퓨터학과	기계학습	육동석	3(3)		
COSE362	02	major_elective	컴퓨터학과	기계학습	김현철	3(3)		
COSE362	03	major_elective	컴퓨터학과	기계학습	강재우	3(3)		
COSE371	01	major_required	컴퓨터학과	데이터베이스	정순영	3(3)		COSE101 COSE102 COSE213
COSE371	02	major_required	컴퓨터학과	데이터베이스	이혁기	3(3)	tuesday 7교시 우정정보관 101 tuesday 8교시 우정정보관 101 thursday 7교시 우정정보관 101	COSE101 COSE102 COSE213

DB에 미리 정의된 admin 계정으로 로그인을 하면 admin을 위한 페이지가 나온다. View All

Courses에서와 같이 모든 과목을 조회하고 검색할 수 있으며, 과목을 추가하거나 과목을 삭제할 수 있는 기능이 추가되어 있다. 과목 추가할 시에 담당교수는 이름이 아닌 고유 번호를 입력해야 한다. 이미 있는 과목을 다시 추가하는 경우에는 과목은 그대로 유지되지만 분반만 추가되도록 구현이 되어 있고, 분반도 이미 있는 경우 강의 시간이 추가될 수 있도록 구현되어 있다. 모든 정보가 일치하는 정보가 이미 있는 경우 오류 메시지가 나온다. 과목 삭제 시에도 없는 과목을 삭제하려고 하는 경우 오류 메시지가 나온다.

2. Schema diagram



3. 사용한 SQL문에 대한 설명

```
drop table takes;
drop table login;
drop table student;
drop table section_time;
drop table section;
drop table instructor;
drop table prereq;
drop table course;
drop table department;
drop table grades;
drop table timeslot;
drop table place;
drop table college;

create table college (
    college_name varchar(10) primary key
);
create table place (
    id serial primary key,
    building varchar(20) not null,
    building_address varchar(4) not null,
    place_name varchar(20)
);
create table timeslot (
    id serial primary key,
    day varchar(9) check(day in ('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday')) not null,
    period int check(period>=1),
    start_time time not null,
    end_time time not null
);
create table grades (
    alphabet varchar(2) primary key,
    number numeric(2, 1) check(number>=0.0) not null
);
create table department (
    dept_name varchar(10) primary key,
    college_name varchar(10) references college(college_name) on delete cascade not null
);
create table course (
    course_id varchar(7) primary key,
    course_name varchar(50) not null,
    dept_name varchar(30) references department(dept_name) on delete cascade not null,
    type varchar(14) check(type in ('major_required', 'major_elective', 'elective')) not null,
    credits int check(credits>=0) not null,
    hour int check(hour>=0) not null
);
```

```

create table prereq (
    course_id varchar(7) references course(course_id) on delete cascade,
    prereq_id varchar(7) references course(course_id) on delete cascade,
    primary key (course_id, prereq_id)
);

create table instructor (
    instructor_id varchar(10) primary key,
    instructor_name varchar(30) not null,
    college_name varchar(10) references college(college_name) on delete cascade not null,
    dept_name varchar(10) references department(dept_name) on delete cascade not null
);

create table section (
    id serial primary key,
    course_id varchar(7) references course(course_id) on delete cascade not null,
    section_id varchar(2),
    year int check(year >= 1905) not null,
    semester varchar(6) check(semester in ('spring', 'summer', 'fall', 'winter')) not null,
    instructor_id varchar(20) references instructor(instructor_id)
);

create table section_time (
    section_id int references section(id) on delete cascade not null,
    timeslot_id int references timeslot(id),
    place_id int references place(id),
    primary key(section_id, timeslot_id)
);

create table student (
    std_id varchar(10) primary key,
    name varchar(20) not null,
    college varchar(10) references college(college_name) on delete cascade not null,
    first_major varchar(10) references department(dept_name) on delete cascade not null,
    second_major varchar(10) references department(dept_name),
    std_instructor varchar(20) references instructor(instructor_id) on delete cascade not null,
    std_status varchar(7) check(std_status in ('present', 'absent', 'kicked')) not null,
    phone_number varchar(11),
    address varchar(100),
    zip_code char(5),
    birthday date not null,
    email varchar(30)
);

create table login (
    std_id varchar(10) references student(std_id) on delete cascade not null,
    id varchar(30) primary key,
    pw varchar(200) not null
);

create table takes (
    std_id varchar(10) references student(std_id) on delete cascade not null,
    section_id int references section(id) on delete cascade not null,
    grade varchar(2) references grades(alphabet),
    primary key (std_id, section_id));

```

데이터베이스 테이블을 구성하는 SQL문이다. 우선 이미 존재하는 테이블이 있을 수 있기 때문에 외래 키 순서를 고려하여 모든 테이블을 삭제한다. 이후 역시 외래 키 순서를 고려하여 테이블을 추가한다.

'college' 테이블은 단과대학의 목록을 저장하는 테이블로, 가변길이문자열 단과대학명을 속성으로 갖고, 그 단과대학명이 기본 키가 된다. 'place' 테이블은 수업 장소의 목록을 저장하는 테이블로, 시리얼(자동증가정수) 고유번호, 가변길이문자열 건물명, 호수, 강의실명을 속성으로 갖고, 고유번호가 기본 키가 된다. 고유번호는 장소가 추가될 때마다 1부터 자동으로 증가하는 정수이다. 건물명과 호수는 NULL을 허용하지 않는다.

'timeslot' 테이블은 수업시각(요일-교시)의 목록을 저장하는 테이블로, 시리얼(자동증가정수) 고유번호, 가변길이문자열 요일, 정수 교시, 시각 시작시각, 종료시각을 속성으로 갖고, 고유번호가 기본 키가 된다. 요일의 경우에는 'monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday'의 문자열만 허용하고 NULL을 허용하지 않으며, 교시의 경우에는 1 이상의 정수만 허용한다. 시작시각과 종료시각은 NULL을 허용하지 않는다.

'grades' 테이블은 부여 가능한 성적의 목록을 저장하는 테이블로, 가변길이문자열 알파벳성적과 고정소수 유리수성적을 속성으로 갖고, 알파벳성적이 기본 키가 된다. 유리수성적은 0.0 이상인 고정소수만 허용하고, NULL을 허용하지 않는다.

'department' 테이블은 학과(전공)의 목록을 저장하는 테이블로, 가변길이문자열 학과명과 단과대학명을 속성으로 갖고, 학과명이 기본 키가 된다. 단과대학명은 'college' 테이블의 단과대학명 속성을 외래 키로 참조하고, 'college' 테이블에서 삭제될 시 같이 삭제되도록 하며, NULL을 허용하지 않는다.

'course' 테이블은 과목의 목록을 저장하는 테이블로, 가변길이문자열 학수번호, 과목명, 개설학과명, 유형, 정수 학점, 주당수업시간을 속성으로 갖고, 학수번호가 기본 키가 된다. 과목명, 개설학과명, 유형, 학점, 주당수업시간은 전부 NULL을 허용하지 않는다. 개설학과명은 'department' 테이블의 학과명을 외래 키로 참조하고, 'department' 테이블에서 삭제될 시 같이 삭제되도록 한다. 유형은 'major_required', 'major_elective', 'elective'의 문자열만 허용하고, 학점과 주당수업시간은 0 이상의 정수만 허용한다.

'prereq' 테이블은 선수과목의 목록을 저장하는 테이블로, 가변길이문자열 학수번호와 선수과목의 학수번호를 속성으로 갖고, 이 두 속성의 조합이 기본 키가 된다. 두 속성은 모두 'department' 테이블의 학수번호를 외래 키로 참조하며, 'department' 테이블에서 삭제될 시 같이 삭제되도록 한다.

'instructor' 테이블은 교수자의 목록을 저장하는 테이블로, 가변길이문자열 고유번호, 교수자명, 소속단과대학명, 소속학과명을 속성으로 갖고, 고유번호가 기본 키가 된다. 교수자명, 소속단과대학명, 소속학과명은 전부 NULL을 허용하지 않는다. 소속단과대학명은 'college' 테이블의 단과대

학명을, 소속학과명은 'department' 테이블의 학과명을 외래 키로 참조하며, 둘 다 참조하는 테이블에서 삭제될 시 같이 삭제되도록 한다.

'section' 테이블은 분반의 목록을 저장하는 테이블로, 시리얼 고유번호, 가변길이문자열 학수번호, 분반코드, 개설학기, 교수자고유번호, 정수 개설연도를 속성으로 갖고, 고유번호가 기본 키가 된다. 학수번호는 'course' 테이블의 학수번호를, 교수자고유번호는 'instructor' 테이블의 고유번호를 외래 키로 참조하고, 학수번호는 'course' 테이블에서 삭제될 시 같이 삭제되도록 하며, NULL을 허용하지 않는다. 개설연도는 1905 이상의 정수만 허용하고, 학기는 'spring', 'summer', 'fall', 'winter'의 문자열만 허용하며, 둘 다 NULL을 허용하지 않는다.

'section_time' 테이블은 분반 별 수업시각의 목록을 저장하는 테이블로, 정수 분반고유번호, 수업시각고유번호, 장소고유번호를 속성으로 갖고, 분반고유번호와 수업시각고유번호의 조합이 기본 키가 된다. 분반고유번호는 'section' 테이블의 고유번호를, 수업시각고유번호는 'timeslot' 테이블의 고유번호를, 장소고유번호는 'place' 테이블의 고유번호를 외래 키로 참조하며, 분반고유번호는 'section' 테이블에서 삭제될 시 같이 삭제되도록 하고 NULL을 허용하지 않는다.

'student' 테이블은 학생의 목록을 저장하는 테이블로, 가변길이문자열 학번, 학생명, 소속단과대학명, 제1전공명, 제2전공명, 지도교수고유번호, 재학상태, 휴대전화번호, 주소, 이메일주소, 고정길이문자열 우편번호, 날짜 생일을 속성으로 갖고, 학번이 기본 키가 된다. 학생명, 소속단과대학명, 제1전공명, 지도교수고유번호, 재학상태, 생일은 전부 NULL을 허용하지 않는다. 소속단과대학명은 'college' 테이블의 단과대학명을, 제1전공명과 제2전공명은 각각 'department' 테이블의 학과명을, 지도교수고유번호는 'instructor' 테이블의 고유번호를 외래 키로 참조하며, 제2전공명을 제외한 나머지 외래 키는 전부 참조하는 테이블에서 삭제될 시 같이 삭제되도록 한다. 재학상태는 'present', 'absent', 'kicked'의 문자열만 허용한다.

'login' 테이블은 사이트의 계정 목록을 저장하는 테이블로, 가변길이문자열 고유식별자(사용자명), SHA512로 암호화된 비밀번호, 학번을 속성으로 갖고, 고유식별자를 기본 키가 된다. 비밀번호와 학번은 둘 다 NULL을 허용하지 않는다. 학번은 'student' 테이블의 학번을 외래 키로 참조하고, 'student' 테이블에서 삭제될 시 같이 삭제되도록 한다.

'takes' 테이블은 수강정보 목록을 저장하는 테이블로, 가변길이문자열 학번, 성적, 정수 분반고유번호를 속성으로 갖고, 학번과 분반고유번호의 조합이 기본 키가 된다. 학번은 'student' 테이블의 학번을, 분반고유번호는 'section' 테이블의 고유번호를, 성적은 'grades' 테이블의 알파벳성적을 외래 키로 참조하며, 성적을 제외한 나머지 외래 키는 둘 다 참조하는 테이블에서 삭제될 시 같이 삭제되도록 하고 NULL을 허용하지 않는다.

또한 프로그램 구동에 사용할 예시 데이터가 적절히 INSERT문을 이용하여 CREATE TABLE 이후에 삽입된다.


```
query = "SELECT s.id AS idx, s.course_id, s.section_id, s.year, s.semester, s.instructor_name, s.college_name, s.dept_name, c.*, temp.* \
FROM (section JOIN instructor ON section.instructor_id=instructor.instructor_id) AS s, \
(course JOIN department ON course.dept_name=department.dept_name) AS c, \
(takes JOIN login ON takes.std_id=login.std_id) AS temp \
WHERE c.course_id=s.course_id AND s.id=temp.section_id AND temp.id= '%s' \
ORDER BY s.year, s.semester, s.course_id, s.section_id ASC;"%user
```

My Courses 페이지에서 수강한 강의의 목록을 불러 오는 질의이다. 분반 정보에 교수자 정보를 교수자고유번호를 기준으로 JOIN하고, 과목 정보에 개설학과 정보를 학과명 기준으로 JOIN하며, 수강 정보에 로그인 정보를 학번 기준으로 JOIN하여 각각을 카르테시안곱을 취하였다. 카르테시안곱의 결과에서 과목과 분반의 학수번호가 일치하고, 분반과 수강 정보의 분반고유번호가 일치하며, 로그인ID가 현재 접속한 사용자인 정보를 SELECT하여 개설연도, 개설학기, 학수번호, 분반코드의 오름차순으로 나열되게 하였다.

```
query2="SELECT * FROM (section_time LEFT JOIN timeslot ON section_time.timeslot_id=timeslot.id) AS temp \
LEFT JOIN place ON temp.place_id=place.id WHERE section_id= '%s' ORDER BY timeslot_id ASC;"%result[i]['idx']
```

My Courses 페이지에서 각각의 강의 정보에 대해서 시간표를 불러 오는 질의이다. 수업시간표에 수업시각을 수업시각고유번호를 기준으로 LEFT JOIN하고, 수업장소를 장소고유번호를 기준으로 LEFT JOIN하여, 분반고유번호가 앞에서 구한 분반고유번호일 때를 SELECT하여 수업시각의 오름차순으로 나열되게 하였다.

```
query3="SELECT prereq_id FROM prereq WHERE course_id= '%s' ORDER BY prereq_id ASC;"%result[i]['course_id']
```

My Courses 페이지에서 마찬가지로 각각의 강의 정보에 대해서 선수과목을 불러 오는 질의이다. 선수과목 목록 중 학수번호가 앞에서 구한 학수번호일 때를 SELECT하여 선수과목의 학수번호의 오름차순으로 나열되게 하였다.

```
query = 'SELECT * FROM (section JOIN instructor ON section.instructor_id=instructor.instructor_id) AS temp1, \
(course JOIN department ON course.dept_name=department.dept_name) AS temp2 WHERE temp2.course_id=temp1.course_id'
```

```

if year != '':
    query += " AND temp1.year=%d" % int(year)
if semester != '':
    query += " AND temp1.semester='%s'" % semester
if college != '':
    if college=='교양':
        query+=" AND temp2.type='elective'"
    else:
        query+=" AND temp2.type<>'elective' AND temp2.college_name='%s'" % college
if credit != '':
    query+=" AND temp2.credits=%d" % int(credit)
if instructor != '':
    query+=" AND temp1.instructor_name LIKE '%%%s%%'" % instructor
if course_id != '':
    query+=" AND temp2.course_id LIKE '%%%s%%'" % course_id
if section_id != '':
    query+=" AND temp1.section_id LIKE '%%%s%%'" % section_id
if course_name != '':
    query+=" AND temp2.course_name LIKE '%%%s%%'" % course_name

```

```

query+=" ORDER BY temp1.year, temp1.semester, temp1.course_id, temp1.section_id ASC;"

```

View All Courses 페이지에서 강의 목록을 불러 오는 질의이다. 분반과 교수자 정보, 과목과 개설학과 정보는 My Courses 페이지와 유사하게 JOIN하였다. 이후 추가적인 검색 질의를 각 요소의 존재 여부에 따라서 제약 조건을 추가하는 방식으로 구현하였다. 교수자명과 학수번호, 분반 코드, 과목명은 LIKE문을 사용하여 부분 일치 검색이 가능하도록 하였다. 마지막으로 개설연도, 개설학기, 학수번호, 분반코드의 오름차순으로 나열되게 하였다.

```

query2="SELECT * FROM (section_time LEFT JOIN timeslot ON section_time.timeslot_id=timeslot.id) AS temp \
LEFT JOIN place ON temp.place_id=place.id WHERE section_id='%s' ORDER BY timeslot_id ASC;" % result[i]['id']

```

View All Courses 페이지에서 각각의 강의 정보에 대해서 시간표를 불러 오는 질의이다. My Courses 페이지에서의 동작과 유사하게 구현하였다.

```

query3="SELECT prereq_id FROM prereq WHERE course_id='%s' ORDER BY prereq_id ASC;" % result[i]['course_id']

```

View All Courses 페이지에서 각각의 강의 정보에 대해서 선수과목을 불러 오는 질의이다. My Courses 페이지에서의 동작과 유사하게 구현하였다.

```

query2 = "SELECT * FROM (section_time LEFT JOIN timeslot ON section_time.timeslot_id=timeslot.id) AS temp \
LEFT JOIN place ON temp.place_id=place.id WHERE section_id='%s'" % i['id']

```

```

if day!='':
    query2+=" AND day='%s'" % day
if time!='':
    query2+=" AND period=%d" % int(time)

```

```

query2+=" ORDER BY timeslot_id ASC;"

```

View All Courses 페이지에서 각각의 강의 정보에 대해서 시간표를 불러 오되, 검색 질의를 추가하여 요일과 교시의 정보를 제한하는 질의이다. 전체적인 구현은 앞의 것과 같으나, 요일 정보와 교시 정보가 있을 경우 조건을 추가하도록 구현하였다.

```
cur.execute(
    "DELETE FROM login WHERE std_id = '%s' AND id='%s';" % (std_id, user)
)
```

My Page 페이지에서 탈퇴하는 질의이다. 'login' 테이블에서 학번과 로그인ID가 일치하는 학생의 계정 정보를 DELETE문을 이용하여 삭제한다.

```
cur.execute(
    "SELECT * FROM student JOIN login ON student.std_id=login.std_id JOIN instructor ON student.std_id=instructor.instructor_id WHERE login.id='%s'" % user
)
```

My Page 페이지에서 개인 정보를 불러 오는 질의이다. 학생 정보에 로그인 정보와 교수자 정보를 JOIN하여 로그인ID가 현재 사용자인 정보를 불러 오도록 SELECT문을 이용하여 불러 온다.

```
cur.execute(
    "SELECT (SUM(credits*number)/SUM(credits))::NUMERIC(3,2) \
FROM (takes LEFT JOIN section ON takes.section_id=section.id) AS temp \
LEFT JOIN course ON course.course_id=temp.course_id LEFT JOIN grades ON temp.grade=grades.alphabet \
JOIN login ON temp.std_id=login.std_id WHERE login.id='%s' AND temp.grade IS NOT NULL;" % user
)
```

My Page 페이지에서 학점 평균을 구하는 질의이다. 수강 정보에 분반 정보를 분반고유번호를 기준으로 LEFT JOIN하고, 과목 정보를 학수번호를 기준으로 LEFT JOIN하고, 성적 정보를 알파벳성적을 기준으로 LEFT JOIN하고, 로그인 정보를 학번을 기준으로 JOIN하였다. 이 JOIN 결과에서 로그인ID가 현재 사용자이고 성적이 NULL이 아닌 강의에 대해서 학점(credit)과 유리수학점(points)를 곱한 값의 합에서 학점(credits)의 합을 나눈 값을 소수점 아래 둘째 자리까지 표현하였다. 합을 구하는 과정에 있어 SUM Aggregation Function을 사용하였다.

```
cur.execute(
    "UPDATE login SET pw='%s' WHERE id = '%s';" % (hashlib.sha512(password.encode()).hexdigest(), id)
)
```

Change Password 페이지에서 비밀번호를 변경하는 질의이다. 'login' 테이블에서 로그인ID가 현재 사용자인 항목을 찾아 SHA512로 암호화된 비밀번호를 UPDATE문을 이용하여 갱신하였다.

```
cur.execute(
    "SELECT * FROM login WHERE id='%s' AND pw='%s';" % (id, hashlib.sha512(password.encode()).hexdigest())
)
```

Login 페이지에서 로그인하는 질의이다. 'login' 테이블에서 로그인ID와 SHA512로 암호화된 비밀번호가 일치하는 항목이 존재하는 경우 로그인에 성공하도록 SELECT문을 이용하여 구현하였다.

```
cur.execute("SELECT COUNT(std_id) FROM student WHERE std_id='%s'" % std_id)
```

Register 페이지에서 학생의 존재 여부를 판단하는 질의이다. 'student' 테이블에서 입력된 학번 정보에 해당하는 학생 정보가 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute("SELECT COUNT(id) FROM login WHERE std_id='%s' OR id='%s'"%(std_id, id))
```

Register 페이지에서 로그인ID 중복 확인을 하는 질의이다. 'login' 테이블에서 입력된 로그인ID에 해당하는 계정 정보가 이미 존재하는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute(
    "INSERT INTO login VALUES ('%s', '%s', '%s');" % (std_id, id, hashlib.sha512(password.encode()).hexdigest())
)
```

Register 페이지에서 회원가입을 하는 질의이다. 'login' 테이블에 학번, 로그인ID, SHA512로 암호화된 비밀번호를 INSERT문을 이용하여 삽입한다.

```
cur.execute("SELECT COUNT(course_id) FROM course WHERE course_id='%s'"%(course_id))
```

Admin 페이지에서 과목 추가 시 이미 있는 과목인지를 확인하는 질의이다. 'course' 테이블에서 입력된 학수번호에 해당하는 과목이 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute(
    "INSERT INTO course VALUES ('%s', '%s', '%s', '%s', '%s', '%s');" % (
        course_id, course_name, dept_name, type, credits, hour)
)
```

Admin 페이지에서 과목을 추가하는 질의이다. 'course' 테이블에 학수번호, 과목명, 개설학과명, 유형, 학점, 시간 정보를 INSERT문을 이용하여 삽입한다.

```
cur.execute("SELECT COUNT(course_id) FROM course WHERE course_id='%s'"%(prereq_id))
```

Admin 페이지에서 과목 추가 시 입력된 선수과목의 학수번호가 존재하는지를 확인하는 질의이다. 'course' 테이블에서 입력된 선수과목의 학수번호가 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute("SELECT COUNT(course_id) FROM prereq WHERE course_id='%s' AND prereq_id='%s'"%(course_id, prereq_id))
```

Admin 페이지에서 과목 추가 시 입력된 선수과목 정보가 이미 기록된 정보인지를 확인하는 질의이다. 'prereq' 테이블에서 입력된 학수번호와 선수과목의 학수번호에 해당하는 항목이 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute("INSERT INTO prereq VALUES ('%s', '%s')"%(course_id, prereq_id))
```

Admin 페이지에서 과목 추가 시 선수과목을 추가하는 질의이다. 'prereq' 테이블에 학수번호,

선수와 과목의 학수번호 정보를 INSERT문을 이용하여 삽입한다.

```
cur.execute("SELECT COUNT(id) FROM section \
WHERE year=%d AND semester='%s' AND course_id='%s' AND section_id='%s'\" \
%(int(year), semester, course_id, section_id))
```

Admin 페이지에서 과목 추가 시 이미 있는 분반인지를 확인하는 질의이다. 'section' 테이블에서 입력된 개설연도, 개설학기, 학수번호, 분반코드에 해당하는 분반이 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute(
"INSERT INTO section VALUES (DEFAULT, '%s', '%s', '%d', '%s', '%s');" \
%(course_id, section_id, int(year), semester, instructor_id)
)
```

Admin 페이지에서 과목 추가 시 분반을 추가하는 질의이다. 'section' 테이블에 학수번호, 분반코드, 개설연도, 개설학기, 교수자고유번호를 INSERT문을 이용하여 삽입한다. 분반고유번호는 시리얼이므로 DEFAULT로 삽입한다.

```
cur.execute("SELECT COUNT(section_time.section_id) FROM section_time JOIN section ON section_time.section_id=section.id \
JOIN timeslot ON section_time.timeslot_id=timeslot.id \
WHERE year=%d AND semester='%s' AND course_id='%s' AND section.section_id='%s' AND day='%s' AND period=%d\" \
%(int(year), semester, course_id, section_id, day, int(time)))
```

Admin 페이지에서 과목 추가 시 입력된 요일, 교시에 해당하는 수업시간을 입력된 분반이 이미 가지고 있는지를 확인하는 질의이다. 강의시간표 정보에 분반 정보를 분반고유번호를 기준으로 JOIN하고, 수업시각 정보를 수업시각고유번호를 기준으로 JOIN하여 입력된 개설연도, 개설학기, 학수번호, 분반코드, 요일, 교시 정보가 일치하는지 분반고유번호가 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute("INSERT INTO section_time \
VALUES ((SELECT id FROM section WHERE year=%d AND semester='%s' AND course_id='%s' AND section_id='%s'), \
(SELECT id FROM timeslot WHERE day='%s' AND period=%d), \
(SELECT id FROM place WHERE building='%s' AND building_address='%s'))\" \
%(int(year), semester, course_id, section_id, day, int(time), building, building_address))
```

Admin 페이지에서 과목 추가 시 강의시간표 정보를 추가하는 질의이다. 'section_time' 테이블에 입력된 개설연도, 개설학기, 학수번호, 분반코드에 해당하는 항목을 Subquery로 찾아 그 분반고유번호와, 'timeslot' 테이블에서 입력된 요일과 교시에 해당하는 항목을 Subquery로 찾아 그 수업시각고유번호와, 'place' 테이블에서 입력된 건물명과 호수에 해당하는 항목을 Subquery로 찾아 그 장소고유번호를 INSERT문을 이용하여 삽입한다.

```
cur.execute("SELECT * FROM section WHERE (course_id, section_id, year, semester)=(('%s', '%s', '%s', '%s'))" \
%(dcourse_id, dsection_id, dyear, dsemester))
```

Admin 페이지에서 강의 삭제 시 입력된 분반 정보에 해당하는 강의가 있는지를 확인하는 질의이다. 'section' 테이블에서 입력된 학수번호, 분반코드, 개설연도, 개설학기에 해당하는 강의가 있는지를 SELECT문과 COUNT Aggregation Function을 이용하여 구하였다.

```
cur.execute(
    "DELETE FROM section WHERE (course_id, section_id, year, semester) = ('%s', '%s', '%s', '%s')" % (dcourse_id, dsection_id, dyear, dsemester)
)
```

Admin 페이지에서 강의를 삭제하는 질의이다. 'section' 테이블에서 입력된 학수번호, 분반코드, 개설연도, 개설학기 정보에 해당하는 강의를 DELETE문을 이용하여 삭제한다.

```
query = 'SELECT * FROM (section JOIN instructor ON section.instructor_id=instructor.instructor_id) AS temp1, \
        (course JOIN department ON course.dept_name=department.dept_name) AS temp2 WHERE temp2.course_id=temp1.course_id'
```

```
if year != '':
    query += " AND temp1.year=%d" % int(year)
if semester != '':
    query += " AND temp1.semester='%s'" % semester
if college != '':
    if college=='교양':
        query+=" AND temp2.type='elective'"
    else:
        query+=" AND temp2.type<>'elective' AND temp2.college_name='%s'%"%college
if credit != '':
    query+=" AND temp2.credits=%d"%int(credit)
if instructor != '':
    query+=" AND temp1.instructor_name LIKE '%s'%"%instructor
if course_id != '':
    query+=" AND temp2.course_id LIKE '%s'%"%course_id
if section_id != '':
    query+=" AND temp1.section_id LIKE '%s'%"%section_id
if course_name != '':
    query+=" AND temp2.course_name LIKE '%s'%"%course_name
```

```
query+=" ORDER BY temp1.course_id, temp1.section_id ASC;"
```

Admin 페이지에서 강의를 검색하는 질의이다. View All Courses 페이지에서의 동작과 유사하게 구현하였다.

```
query2="SELECT * FROM (section_time LEFT JOIN timeslot ON section_time.timeslot_id=timeslot.id) AS temp \
        LEFT JOIN place ON temp.place_id=place.id WHERE section_id='%s' ORDER BY timeslot_id ASC;"%result[i]['id']
```

Admin 페이지에서 각각의 강의 정보에 대하여 시간표를 불러 오는 질의이다. 앞에서의 동작과 유사하게 구현하였다.

```
query3="SELECT prereq_id FROM prereq WHERE course_id='%s' ORDER BY prereq_id ASC;"%result[i]['course_id']
```

Admin 페이지에서 각각의 강의 정보에 대하여 선수과목을 불러 오는 질의이다. 앞에서의 동작과 유사하게 구현하였다.

```
query2 = "SELECT * FROM (section_time LEFT JOIN timeslot ON section_time.timeslot_id=timeslot.id) AS temp \
        LEFT JOIN place ON temp.place_id=place.id WHERE section_id='%s' % i['id']
```

```
if day!='':
    query2+=" AND day='%s'"%day
if time!='':
    query2+=" AND period=%d"%int(time)
```

```
query2+=" ORDER BY timeslot_id ASC;"
```

Admin 페이지에서 각각의 강의 정보에 대해서 시간표를 불러 오되, 검색 질의를 추가하여 요일과 교시의 정보를 제한하는 질의이다. View All Courses 페이지에서의 동작과 유사하게 구현하였다.

```
query = 'SELECT * FROM (section JOIN instructor ON section.instructor_id=instructor.instructor_id) AS temp1, \
(course JOIN department ON course.dept_name=department.dept_name) AS temp2 WHERE temp2.course_id=temp1.course_id'
query+=" ORDER BY temp1.course_id, temp1.section_id ASC;"
```

Admin 페이지에서 전체 강의 목록을 불러 오는 질의이다. 앞의 것과 유사하게 구현하였으나, 검색 질의에 따라 추가적인 조건을 덧붙이지 않는다.

```
query2="SELECT * FROM (section_time LEFT JOIN timeslot ON section_time.timeslot_id=timeslot.id) AS temp \
LEFT JOIN place ON temp.place_id=place.id WHERE section_id='%s' ORDER BY timeslot_id ASC;"%result[i]['id']
```

Admin 페이지에서 각각의 강의 정보에 대해서 시간표를 불러 오는 질의이다. 앞에서의 동작과 유사하게 구현하였다.

```
query3 = "SELECT prereq_id FROM prereq WHERE course_id= '%s' ORDER BY prereq_id ASC;" % result[i]['course_id']
```

Admin 페이지에서 각각의 강의 정보에 대해서 선수과목을 불러 오는 질의이다. 앞에서의 동작과 유사하게 구현하였다.