데이터 베이스

컴퓨터공학과 19011547 김상원

[주제: 대학교 수강신청 사이트]

[제출일: 2023-06-23]

목차

1. 주제 선정

간단한 대학교 수강 신청 사이트를 만들었습니다. 해당 프로그램의 기능은 크게 두 가지로 구분할 수 있습니다. 첫 번째는 수강 신청과 취소 기능입니다. 두 번째는 구성원 조회 기능입니다. 권한에 높은 'director'로 접근하면 두 가지 기능 모두 수행할 수 있지만 권한이 낮은 'selector'로 접근하면 구성원 조회 기능만 수행할 수 있습니다.

현실감 있는 프로그램을 만들기 위해 '세종대학교 학사정보시스템'에서 '컴퓨터공학과', '인공지능학과', '소프트웨어학과', '전자정보통신공학과'의 23년 1학기 시간표를 가져왔습니다. 이를 통해 274개의 강의 시간표를 확보하여 보다 현실감 있는 프로그램을 작성할 수 있었습니다. 또한 이번 학기에 배운 SQL문을 최대한 활용하는 것에 초점을 두었습니다.

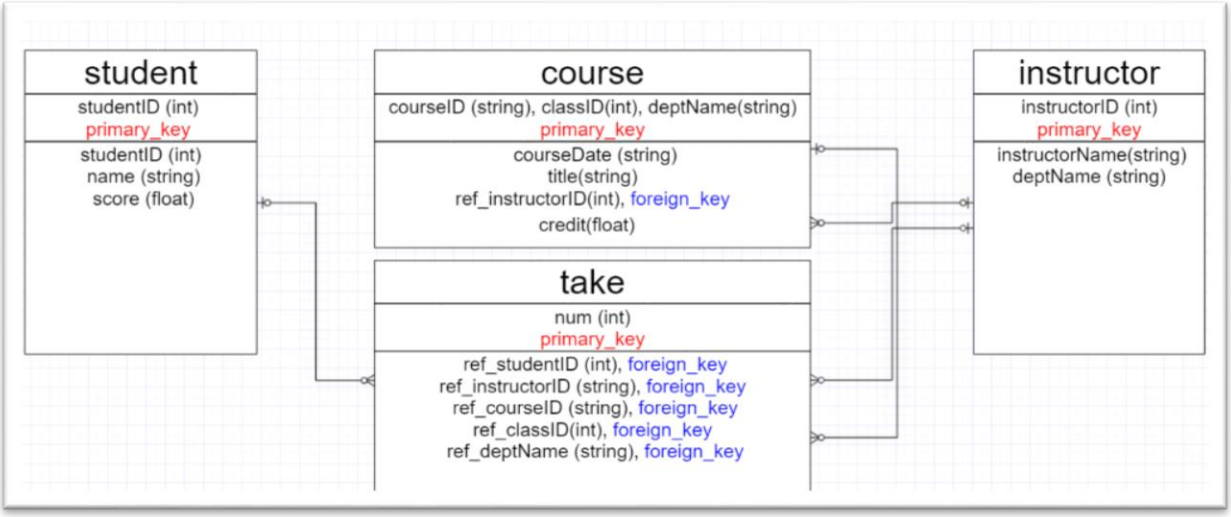
2. 제약사항

첫 번째 기능인 수강 신청, 취소 기능을 수행하기 위해서는 몇 가지 제약 사항이 존재합니다. 우선 수강 신청 기능(register)을 수행할 때의 제약 사항은 크게 3가지입니다. 첫번째는 한 학생 당 18학점(credit)까지 수강 신청할 수 있습니다. 두 번째는 학수 번호 기준으로 동일한 과목을 수강하면 안됩니다. 세 번째는 수강 신청한 수업이 서로 겹치(overlap)는 시간대가 있으면 안됩니다. 위의 세가지의 규칙을 만족할 때만 수강 신청을 할 수 있습니다. 그리고 수강 취소 기능(drop)을 수행할 때는 수강 취소하기 이전에 수강 신청을 했어야 합니다. 위의 규칙을 만족할 때만 수강 취소를 할 수 있습니다.

두 번째 기능인 조회 기능을 수행할 때는 항상 조회는 가능하나, 조회하고자 하는 값이 데이터베이스에 존재하지 않으면 어떠한 값도 조회되지 않습니다.

위의 두 가지 기능 정상적으로 수행이 완료되면 tuple형태로 구성된 True가 반환되고, 실패하면 False가 반환됩니다. 이를 통해 코드 작성자는 명령어의 정상 완료 여부를 확인할 수 있습니다.

3. 테이블 구조와 Entity Relationship Diagram



위의 그림은 제가 제출한 폴더 내의 make_table.sql로 생성한 테이블의 관계도 입니다. project라는 데이터베이스 내에는 student, course, instructor, take라는 4개의 테이블이 존재합니다.

student 테이블은 학생에 관한 데이터를 갖고 있습니다. 학번(studentID)이 primary key이고 학생 이름(name), 학생의 성적(score)로 구성됩니다.

instructor 테이블은 교수님들에 관한 데이터를 갖고 있습니다. 교수님 고유 번호(instructorID)가 primary key이고 교수님 성함(instructorName)과 교수님 소속학과(depName)으로 구성됩니다.

course 테이블은 강의에 관한 데이터를 갖고 있습니다. 학수번호(courseID), 분반(classid), 소속학과(deptName)으로 primary key를 구성하고 강의 시간표(courseDate), 강의 이름(title), 교수님의 고번호(ref_instructor), 학점(credit)으로 구성되며 ref_instructor는 instructor 테이블과 foreign key와 primary key 관계입니다.

take 테이블은 수강 신청 내역에 관한 데이터를 갖고 있습니다. 수강 신청 내역 고유 번호(num)를 primary key이고 학번(ref_studentID), 교수님 고유번호(ref_instructorID), 학수번호(ref_courseID), 분반(ref_class), 학과(ref_depName)으로 구성되어 있습니다. ref_studentID는 student table의 studentID와, ref_instructorID는 instructor table의 instrctorID와, ref_courseID, ref_classID, ref_depName은 course의 courseID, classID, depName과 foreign key와 primary key 관계입니다.

4. 데이터 전처리

preprocessing폴더 내의 preprocessing.py로는 INSTRUCTOR TABLE, COURSE TABLE을 생성하기 위한 데이터를 전처리하여 생성합니다.

```
# 인공지능학과, 소프트웨어학과, 컴퓨터공학과, 전자정보통신학과의 23-1 강의시간표를 하나의 exel로 합친 후 read하였습니다.
data = pd.read_excel('19011547_김상원_DB\data\sejong_univ_data.xlsx')
print(data.columns)

# course에 대한 data를 생성합니다.
preprocessed_data = data[['학수번호', '분반', '교과목명', '교수명', '주관학과', '요일 및 강의시간', '학점/이론/실습']]
```

```
42  # PROJECT.INSTRUCTOR TABLE
43  instructor_data = preprocessed_data[['instructorID', 'instructorName', 'deptName']]
44  # print("this=>\n", instructor_data.columns)
45  instructor_data = instructor_data.drop_duplicates(subset=[('instructorID',)])
46  instructor_data.to_csv('19011547_김상원_DB\data\instructor_data.csv', index = False)
47
48  # PROJECT.COURSE TABLE
49  preprocessed_data = preprocessed_data.rename(columns = {'instructorID': 'ref_instructorID'})
50  course_data = preprocessed_data[['courseID', 'classID', 'title', 'deptName', 'courseDate', 'ref_instructorID', 'credit']]
51  course_data.to_csv('19011547_김상원_DB\data\course_data.csv', index = False)
52
```

위의 코드처럼 4개의 학과를 하나로 묶은 sejong_univ_data.xlsx를 pandas로 변환하고 columns 중에서 필요한 columns들만 preprocessed_data에 담았습니다. 또한 일련의 전처리 과정을 거친 후에 위와 같은 INSTRUCTOR TABLE, COURSE TABLE를 생성하였습니다.

make_student.py로는 임의로 생성한 student_name, student_id, student_grade로 STUDENT TABLE을 생성하기 위한 데이터를 생성합니다.

```
student = pd.DataFrame({'student_name': student_name, 'studentID': student_id, 'student_grade':score})
print(student)
student.to_csv('19011547_김상원_DB\data\student_data.csv', index = False)
```

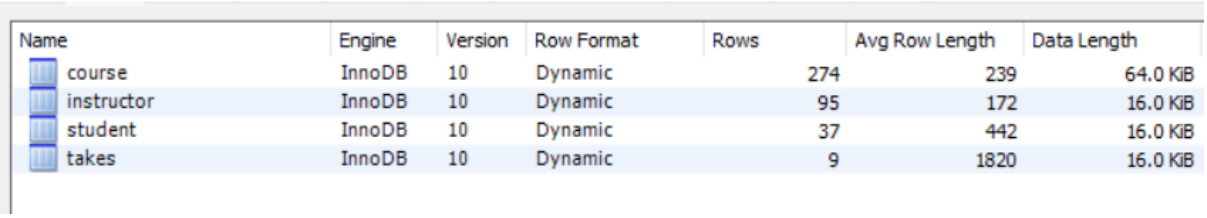
5. SQL과 Python을 통한 데이터베이스 구축

우선 make_table.sql을 통해 table을 생성합니다. 아래의 코드는 make_table.sql문에서 project라는 데이터베이스부터 생성하는 코드임을 확인할 수 있습니다. 또한 해당 코드의 결과를 mysql을 통해 확인할 수 있습니다.

```sql
-- Schema  PROJECT
DROP SCHEMA IF EXISTS PROJECT;

-- Schema  PROJECT
CREATE SCHEMA IF NOT EXISTS PROJECT DEFAULT CHARACTER SET utf8 ;
USE PROJECT;

```



Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length
course	InnoDB	10	Dynamic	274	239	64.0 KiB
instructor	InnoDB	10	Dynamic	95	172	16.0 KiB
student	InnoDB	10	Dynamic	37	442	16.0 KiB
takes	InnoDB	10	Dynamic	9	1820	16.0 KiB

insert_data.py에 있는 아래의 함수를 통해 전처리 과정으로 생성한 course_data.csv, instructor_data.csv, student_data.csv를 각각의 테이블에 insert합니다.

```python
# table_name과 data_name을 통해 table에 data를 insert하였습니다.
def insert_query(cursor, table_name, data_name):
    # insert_query
    df = pd.read_csv('19011547_김상원_DB\data\\' + data_name)
    columns_name = df.columns.tolist()
    print(columns_name)
    columns = df.shape[1]
    rows = df.shape[0]

    query_columns = ', '.join(['%s'] * columns)
    for i in range(rows):
        result = df.iloc[i].astype(str)
        row_data = tuple(result)
        print(result)
        insert_query = f"INSERT INTO {table_name} ({', '.join(columns_name)}) VALUES ({query_columns})"

        print(row_data, "\n")
        print(insert_query, "\n")
        cursor.execute(insert_query, row_data)
```

6. SQL과 Python을 통한 함수 구현

다음은 register.py에 위치한 register함수에 대한 설명입니다. 해당 함수는 크게 4가지 영역으로 구분되어 있습니다.

첫 번째 영역은 instructor_id를 구하는 함수입니다. 사용자는 instructor_id를 알기 어렵기 때문에 교수님의 성함과 학과 이름을 통해 INSTRUCTOR 테이블에서 찾아서 반환합니다.

```
42  def register(cursor, value):
43      # print('입력: studentId, instructorName, deptName, course_id, class_id')
44      # studentId, instructorName, deptName, courseId, classId = input().split()
45      value = [v.strip() for v in value]
46      studentId, instructorName, deptName, courseId, classId = value[0], value[1], value[2], value[3], value[4]
47      instructorId = search_instructor_id(cursor, instructorName, deptName)[0]
48      print("instructorId=>", instructorId)
```

```
# 교수님에 대응되는 id를 반환한다.
def search_instructor_id(cursor, name, dept):
    print("search_instructor_id:", name, dept)
    query = "SELECT INSTRUCTOR.instructorID FROM INSTRUCTOR \
        WHERE INSTRUCTOR.instructorName=%s and INSTRUCTOR.deptName=%s"
    cursor.execute(query, (name, dept))
    instruction_id = cursor.fetchone()
```

두 번째 영역은 학점을 확인하는 영역입니다. 첫번째 query를 통해 현재 수강하려는 과목의 학점을 반환하고 두 번째 query를 통해 해당 학생이 지금까지 수강한 과목의 학점을 반환합니다. 두 번의 query를 통해 얻은 값의 18학점 초과 유무를 검사합니다.

```
51      # CREDIT이 18학점을 넘으면 안된다. (1)
52      # 현재 수강하려고 하는 과목의 credit을 select한다.
53      query = """
54      SELECT COURSE.credit
55      FROM COURSE
56      WHERE COURSE.courseID=%s AND COURSE.classID=%s AND COURSE.deptName=%s
57      """
58      cursor.execute(query, (courseId, classId, deptName, ))
59      cur_credit = cursor.fetchone()[0]
60
61      # 지금까지 학생이 수강한 과목의 총 credit 더해서 select한다.
62      query = """
63      SELECT SUM(COURSE.credit)
64      FROM TAKES
65      INNER JOIN COURSE
66      ON TAKES.ref_courseID=COURSE.courseID AND TAKES.ref_classID=COURSE.classID AND TAKES.ref_deptName=COURSE.deptName
67      WHERE TAKES.ref_studentID=%s
68      """
69      print(cur_credit)
70      cursor.execute(query, (studentId, ))
71      courseCredit = cursor.fetchone()[0]
72      if courseCredit == None:
73          query = "INSERT INTO TAKES (ref_studentID, ref_instructorID, ref_courseID, ref_classID, ref_deptName) VALUES (%s, %s, %s, %s, %s)"
74          cursor.execute(query, (studentId, instructorId, courseId, classId, deptName, ))
75          cursor.fetchall()
76          return True
```

세번째 영역은 해당 과목을 수강 신청 여부를 검사하는 코드입니다.

```
83
84      # 같은 과목 재수강 허용안하는 쿼리 (2)
85      # 같은 과목의 기준은 학수번호의 동일함이다.
86
87      query = """
88      SELECT COUNT(*) AS row_count
89      FROM TAKES
90      WHERE TAKES.ref_courseID=%s AND TAKES.ref_studentID=%s
91      """
92      cursor.execute(query, (courseId, studentId))
93      overlap = cursor.fetchone()[0]
94      if overlap == None:
95          overlap = 0
96      print('overlap:\n', overlap)
97      if overlap > 0:
98          print('your course is overlap')
99          return False
100
```

네 번째 영역은 시간대를 확인하는 코드입니다. 겹치는 지를 확인합니다.

```
100
101     # 중복되는 시간대의 수업을 못 듣는 쿼리 (3)
102     # 자신이 듣고 싶은 수업의 시간대를 받는다.
103     query = """
104     SELECT COURSE.courseDate
105     FROM COURSE
106     WHERE COURSE.courseID=%s AND COURSE.classID=%s AND COURSE.deptName=%s
107     """
108     cursor.execute(query, (courseId, classId, deptName, ))
109     cur_date = []
110     cur_date.append(cursor.fetchone()[0])
111     if cur_date[0] == '없음': # 예외처리
112         cur_date[0] = '없00:00-00:00'
113     print(cur_date)
114     # 지금까지 신청한 수업의 시간대를 받는다.
115     query = """
116     SELECT COURSE.courseDate
117     FROM TAKES
118     INNER JOIN COURSE
119     ON TAKES.ref_courseID=COURSE.courseID AND TAKES.ref_classID=COURSE.classID AND TAKES.ref_deptName=COURSE.deptName
120     WHERE TAKES.ref_studentID=%s
121     """
122     cursor.execute(query, (studentId, ))
123     date = cursor.fetchall()
124     date = two2oneArray(date)
125
126     print("date:", date)
127     # 지금까지 신청한 수업의 시간대와 듣고자 하는 시간대를 비교한다.
```

```
154
155     # (1), (2), (3) 비교하고 INSERT한다.
156     # studentId, instructorName, deptName, courseId, classId
157     query = "INSERT INTO TAKES (ref_studentID, ref_instructorID, ref_courseID, ref_classID, ref_deptName) VALUES (%s, %s, %s, %s, %s)"
158     cursor.execute(query, (studentId, instructorId, courseId, classId, deptName, ))
159     take_count = cursor.fetchall()
160     return True
```

다음은 other_utility.py에 위치한 함수들에 대한 설명입니다. 해당 파일에는 비교적 간단한 수강 취소 함수(drop)와 4개의 탐색 관련 함수들로 구성되어 있습니다.

drop함수는 우선 이전에 해당 과목을 신청한 지를 확인한 후에 내부적으로 drop 전후를 비교하고, 해당 과목을 drop합니다.

```
78  def drop(cursor, value):
79      # print('입력: 학수번호와 학번을 입력하세요.')
80      # courseId, studentId = input().split()
81      value = [v.strip() for v in value]
82      courseId, studentId = value[0], value[1]
83      # drop하기 전에 해당 과목을 신청했는지 확인합니다.
84      query = "SELECT COUNT(*) FROM TAKES WHERE ref_courseID=%s and ref_studentID=%s"
85      cursor.execute(query, (courseId, studentId, ))
86      take_count = cursor.fetchone()[0]
87      if take_count == 0:
88          print("drop fail\n")
89          return False
```

```
111
112     # drop한다.
113     query = "DELETE FROM TAKES WHERE ref_studentID=%s and ref_courseID=%s"
114     cursor.execute(query, (studentId, courseId, ))
115     print("drop successfully")
116     return True
```

나머지 Search함수들은 논리적으로 유사하고 간단하기 때문에 각 함수의 기능만 간략하게 소개하겠습니다. 우선 search_instructor_course()는 학수 번호가 동일한 수업들을 하시는 교수님들의 성함을 반환합니다. search_student_course()는 학수 번호가 동일한 수업들을 수강하는 학생들의 명단을 반환합니다. search_score_course()는 학수 번호가 동일한 수업들을 수강하는 학생들의 평균 학점과 인원 수를 반환합니다. search_student_has()는 특정 학번의 학생이 수강 신청한 수업들을 반환합니다.

```
60
61  def search_student_has(cursor, value): # 현재 학생이 수강 신청한 수업을 출력한다.
62      studentId = value.strip()
63      query = """
64      SELECT COURSE.title
65      FROM TAKES
66      INNER JOIN COURSE
67      ON TAKES.ref_courseID=COURSE.courseID AND TAKES.ref_classID=COURSE.classID AND TAKES.ref_deptName=COURSE.deptName
68      WHERE TAKES.ref_studentID=%s
69      """
70      cursor.execute(query, (studentId, ))
71      course_name = cursor.fetchall()
```

```
43  def search_score_course(cursor, value): # course를 수강하는 학생들의 평균 학점 및 인원 수
44      # print('입력: 학수번호를 입력하세요.')
45      # courseId = input().split()
46      courseId = value.strip()
47      query = """
48      SELECT COUNT(*), AVG(STUDENT.student_grade)
49      FROM TAKES
50      INNER JOIN STUDENT
51      ON TAKES.ref_studentID=STUDENT.studentID
52      WHERE TAKES.ref_courseID=%s
53      """
54      cursor.execute(query, (courseId, ))
55      count, score = cursor.fetchone()
```

```
23
24  def search_student_course(cursor, value): # course를 수강하는 학생들 명단 출력
25      # print('입력: 학수번호를 입력하세요.')
26      # courseId = input().split()
27      courseId = value.strip()
28      query = """
29      SELECT STUDENT.student_name
30      FROM TAKES
31      INNER JOIN STUDENT
32      ON TAKES.ref_studentID=STUDENT.studentID
33      WHERE TAKES.ref_courseID=%s
34      """
35      cursor.execute(query, (courseId, ))
36      names = cursor.fetchall()
```

```python
def search_instructor_course(cursor, value): # course를 수업하시는 교수님 명단 출력
    # print('입력: 학수번호를 입력하세요.')
    # courseId = input().split()
    # print(courseId, "\n\n")
    courseId = value.strip()
    query = """
    SELECT INSTRUCTOR.instructorName
    FROM COURSE
    INNER JOIN INSTRUCTOR
    ON COURSE.ref_instructorID=INSTRUCTOR.instructorID
    WHERE COURSE.courseID=%s
    """
    cursor.execute(query, (courseId, ))
    names = cursor.fetchall()
```

7. FLASK를 통한 웹서버 구축

index.html 내에서 name, password를 입력받아 log_in dictionary에 저장한 후에 /main에서 mysql에 접속할 때 사용하빈다. 해당 name, password에 따라 권한이 달라집니다. director는 모든 권한을 갖고 있으나 selector는 select 즉 탐색만 할 수 있습니다.

```python
log_in = {}
@app.route('/', methods=['GET', 'POST'])
def ft_main():
    if request.method == 'POST':
        name = request.form['name']
        password = request.form['password']
        log_in['name'] = name
        log_in['password'] = password
        print(name, password)
        return redirect('/main')
    return render_template('index.html')
```

아래의 코드에서 post형식으로 값을 받아서 활용합니다. main.html에서 값을 받은 후에 이를 try-catch문으로 오류 검사를 수행한 후에 적절한 SQL명령문을 수행합니다.

```python
@app.route('/main', methods=['GET', 'POST'])
def ft_query():
    result = []
    selected_sql = ""
    if request.method == 'POST':
        register_value = request.form.getlist('input1')
        drop_value = request.form.getlist('input2')
        Instructor_value = request.form['input3']
        Student_value = request.form['input4']
        SearchScore = request.form['input5']
        SearchStudentHas = request.form['input6']
        # print(Student_value)
```

```
            selected_sql = 'having'
            result = other_utility.search_student_has(cursor, SearchStudentHas)
        dbConn.commit()
        cursor.close()
        dbConn.close()
        print('MySQL Connection Close')
    except mysql.connector.Error as e:
        print('Database connecting Error: ',e)
        print("Error:\n\n", selected_sql, result)
        return 'Fail'
    if result[-1] == False:
        return 'Fail in function'
    result = result[:-1]
    return render_template('result.html', my_array = result, sql = selected_sql)
return render_template('main.html')
```