데이터베이스 시스템 설계 리포트

20153598

신동민

1. 제목 : 학사생활도우미
2. 소개 : 강의, 학생 정보를 나타내는 DB를 자유자재로 검색 및 변경함으로써 학교 생활에 도움을 준다.
3. 개발환경, DBMS

:

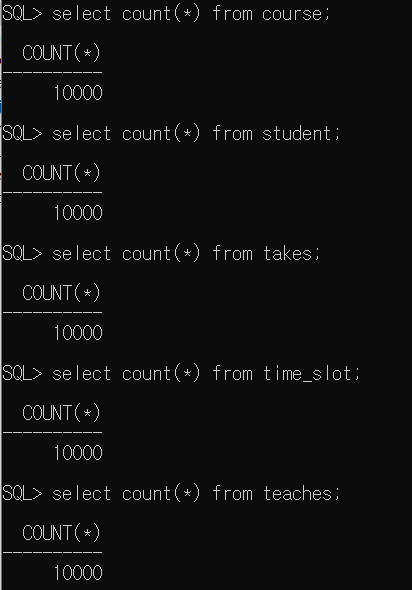
OS: Windows 10

DMBS: Oracle Database 18c Express Edition

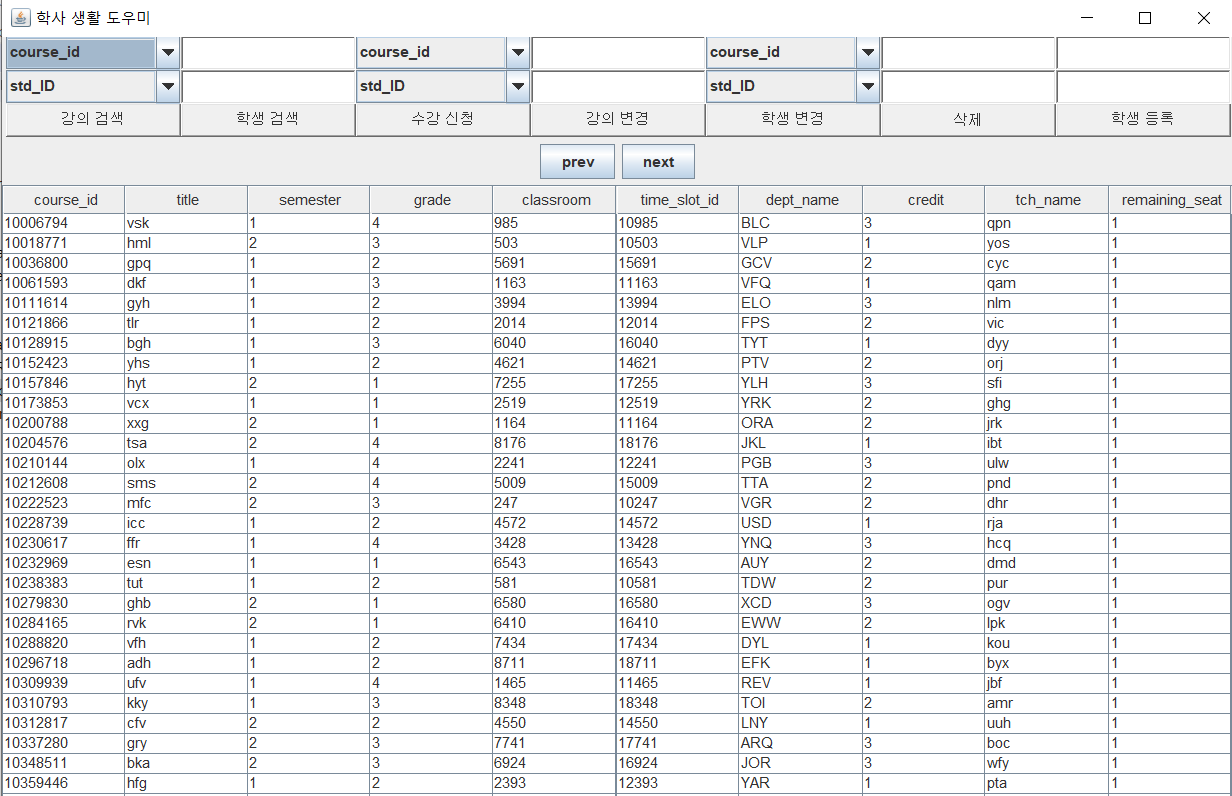
대화식 SQL 도구: Oracle DB – SQL Plus

JAVA eclipse - JDBC

1. 관계 DB 스키마 : “관계 DB 스키마.pdf” 로 첨부.
2. Create table 문 : “create\_table.rtf” (워드 패드) 로 첨부
3. Load한 레코드 수 : 5개 테이블에 각 1만 개



1. 프로그램 GUI

:

1. GUI 설명 및 응용의 기능 개요

: 크게 총 네 개의 영역으로 나뉜다.

내림차순으로, 첫 번째 줄은 강의와 관련된 기능을 하며, 두 번째 줄은 학생과 관련된 기능을 한다. 그리고 세 번째 줄은 페이지를 이동하기 위한 버튼들이며, 네 번째 줄은 검색 결과다. 이것은 첫 화면에는 course\_id 에 의해 정렬된 course 테이블의 레코드를 보여준다.

먼저 첫 번째 줄의 combobox는 course 테이블의 총 10개의 칼럼 ( course\_id, title, semester, grade, classroom, time\_slot\_id, dept\_name, credit, tch\_name. remaining\_seat ) 을 가리키며, 선택된 칼럼에 대한 텍스트를 받기 위한 텍스트 필드가 각각 마련되어 있다. 마지막 텍스트 필드는 값 변경 시 사용되는 텍스트 필드다.

마찬가지로, 두 번째 줄의 세 개의 combobox는 student 테이블의 총 3개의 칼럼 ( std\_ID, std\_name, dept\_name ) 을 가리키며, 선택된 칼럼에 대한 텍스트를 받기 위한 텍스트 필드가 각각 마련되어 있다. 마지막 텍스트 필드는 값 변경 시 사용되는 텍스트 필드다.

보다 효율적인 설명을 위해 각 combobox와 text field 들의 번호를 지정하겠다.

첫 번째 줄의 combobox인 CB\_1 은 순서대로 CB\_1\_1, CB\_1\_2, CB\_1\_3이며, text filed 인 TF\_1은 순서대로 TF\_1\_1, TF\_1\_2, TF\_1\_3, TF\_1\_4이다.

두 번째 줄의 combobox 인 CB\_2 은 순서대로 CB\_2\_1, CB\_2\_2, CB\_2\_3이며, text filed 인 TF\_2은 순서대로 TF\_2\_1, TF\_2\_2, TF\_2\_3, TF\_2\_4이다.

한편 세 번째 영역에는 {강의 검색, 학생 검색, 수강 신청, 강의 변경, 학생 변경, 삭제, 학생 등록} 버튼들이 있다.

1. 구현 트랜잭션의 기능별 분류

총 9개 (T1 ~ T9)의 트랜잭션 중 7개 (T5, T9 제외) 구현 성공

검색 트랜잭션 2개 : T1 (강의 검색), T2 (학생 검색) – 구현 성공

변경 트랜잭션 2개 : T7 (강의 변경), T6 (학생 등록) – 구현 성공

혼합 트랜잭션 2개 : T5 (수강 신청 취소), T4 (학생 삭제) – T4만 구현 성공

Be 혼합 트랜잭션 2개 : T3 (강의 삭제), T8 (학생 변경) – 구현 성공

ABCDE 혼합 트랜잭션 1개 : T9 (수강 신청) – 구현 실패

1. 기능, 트랜잭션, 구현 요건 충족, 소스 코드 및 프로그램 화면 캡처 및 설명
2. 강의 검색
3. 트랜잭션 설계

강의 검색은 트랜잭션 T1에 해당하며 course table의 총 10개의 칼럼에서 1~3개의 키워드를 통해 검색 결과를 제공한다.

CB\_1 과 TF\_1 을 사용하며, CB\_1 에서 칼럼을 선택할 수 있고, TF\_1 에 각기 그에 맞는 키워드를 넣을 수 있다.

키워드를 넣지 않고 클릭 시 course\_id에 의해 내림차순 정렬된 course 테이블의 레코드를 보여준다.

1. 구현 요건 충족

구현 요건 : 복수의 검색 ONLY SQL.

버튼을 눌러 트랜잭션 실행 시,

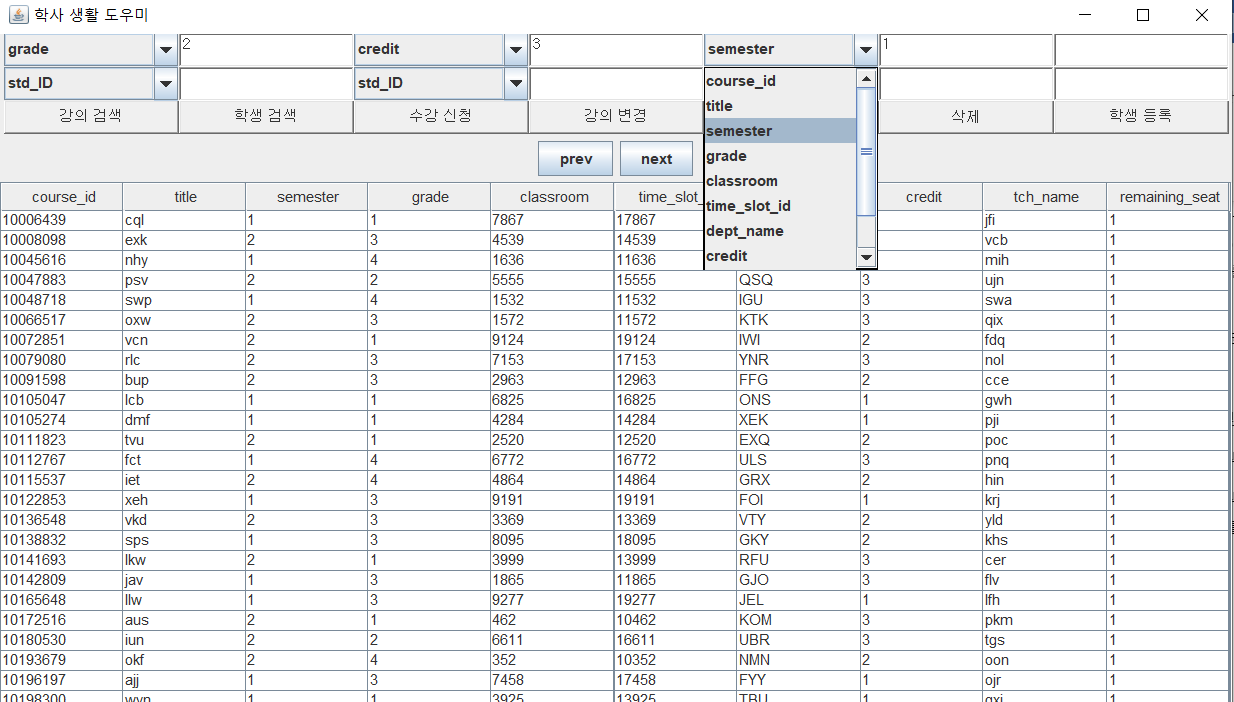
SQL문 S1-1 = “select \* from course” 을 먼저 실행.

그 후, 입력되는 키워드의 수에 따라

SQL문 S-2 를 실행.

1. 실행화면

예시는 grade = 2, credit = 3, semester = 1 에 해당하는 course 테이블 내 레코드들을 보여준다.



1. 소스 코드

UI로부터 칼럼 index 와 키워드를 각각 inx, item 라는 매개변수로 받아온다.

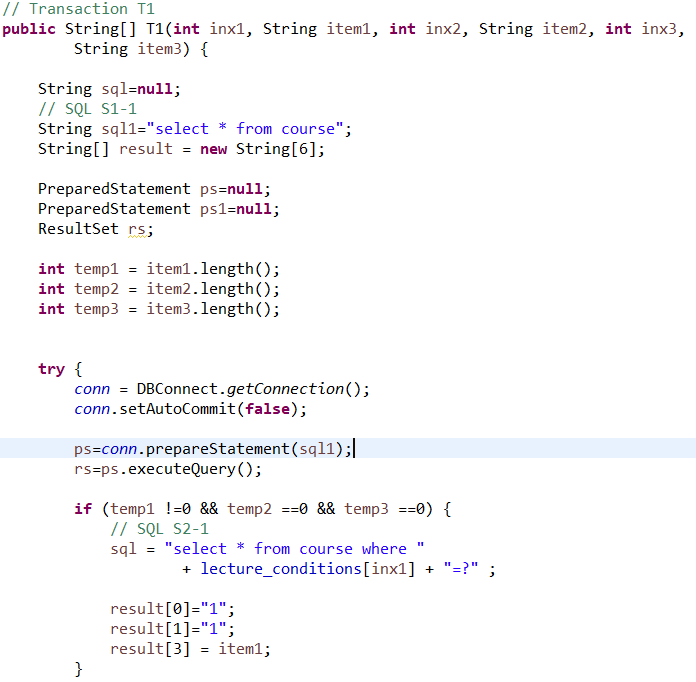
먼저 “select \* from course” 인 SQL문 S-1을 실행하여 테이블을 course 테이블의 모든 레코드를 보이도록 초기화시킨 다음,

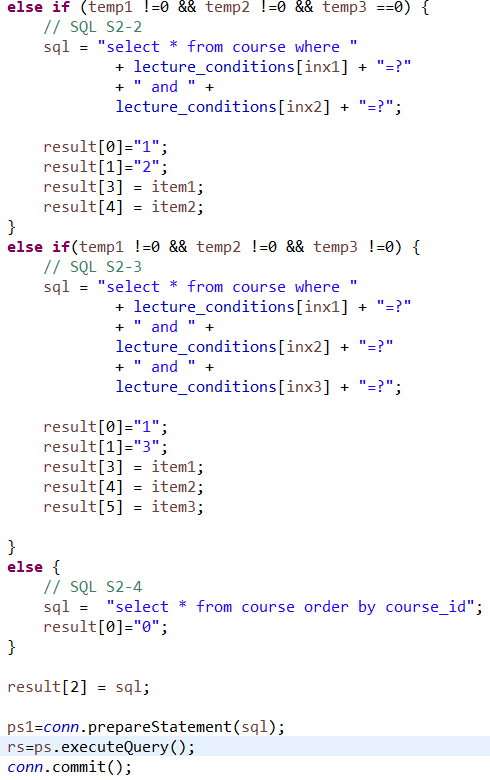
매개변수로 받은 index와 키워드의 개수에 따라 동적 SQL문을 다르게 설정한다.

다시 말해, 키워드가 한 개이면 SQL문 S2-1을, 키워드가 두 개이면 SQL문 S2-2를, 키워드개 세 개이면 SQL문 S2-3이라는 동적 SQL문을 설정한다.

가령, 위의 예시의 실행 화면에서는, 키워드 3개를 받아, SQL문 S2-3 = “select \* from course where grade = 2, credit = 3, semester = 1”을 구성한다.

그 후 commit한다.





1. 학생 검색
2. 설명

학생 검색은 트랜잭션 T2에 해당하며 student table의 총 3개의 칼럼에서 1~3개의 키워드를 통해 검색 결과를 제공한다. 키워드를 넣지 않고 클릭 시 student 테이블의 레코드를 보여준다.

CB\_2 과 TF\_2 을 사용하며, CB\_2 에서 칼럼을 선택할 수 있고, TF\_1-2 에 각기 그에 맞는 키워드를 넣을 수 있다.

키워드를 넣지 않고 클릭 시 std\_id 에 의해 내림차순 정렬된 student 테이블의 레코드를 보여준다.

1. 구현 요건 충족

구현 요건 : 복수의 검색 ONLY SQL.

버튼을 눌러 트랜잭션 실행 시,

SQL문 S1-1 = “select \* from student” 을 먼저 실행.

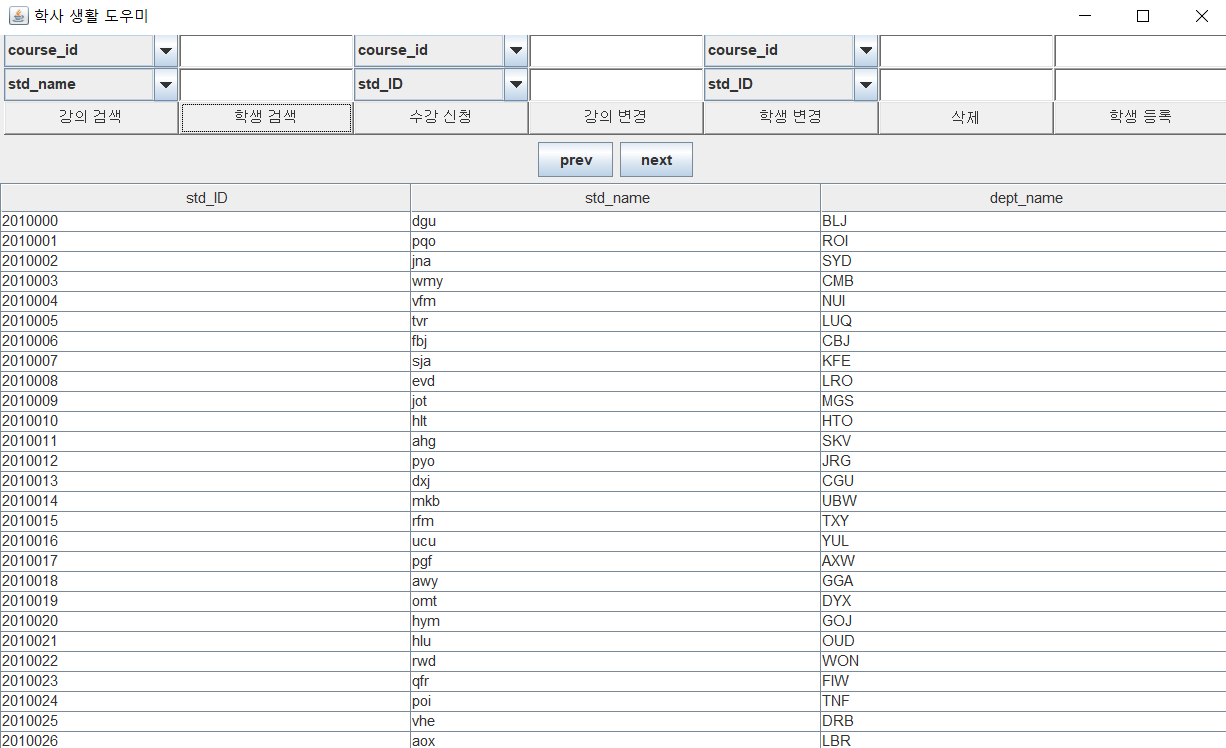
그 후, 입력되는 키워드의 수에 따라

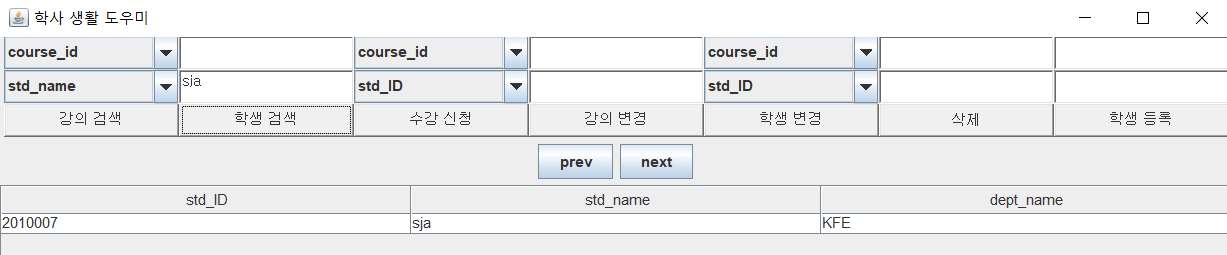
SQL문 S-2 를 실행.

1. 실행화면

첫 번째 예시는 검색어 없이 오직 학생 검색 버튼만 눌렀을 경우이다.

두 번째 예시는 std\_name = sja 에 해당하는 sutdent 테이블 내 레코드들을 보여준다.





1. 소스 코드

UI로부터 칼럼 index 와 키워드를 각각 inx, item 라는 매개변수로 받아온다.

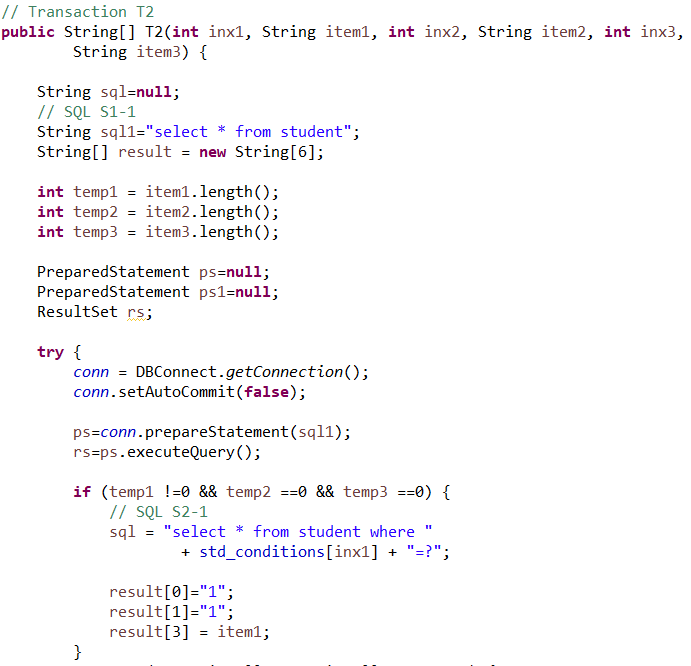
먼저 “select \* from student” 인 SQL문 S-1-1을 실행하여 테이블을 student 테이블의 모든 레코드를 보이도록 초기화시킨 다음,

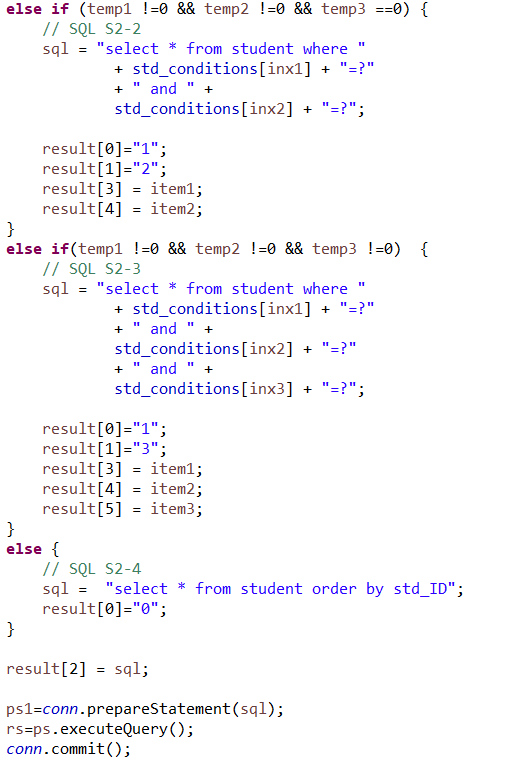
매개변수로 받은 index와 키워드의 개수에 따라 동적 SQL문을 다르게 설정한다.

다시 말해, 키워드가 한 개이면 SQL문 S2-1을, 키워드가 두 개이면 SQL문 S2-2를, 키워드개 세 개이면 SQL문 S2-3이라는 동적 SQL문을 설정한다.

가령, 위의 예시의 실행 화면에서는, 키워드 1개를 받아, SQL문 S2-1 = “select \* from student where std\_name = ‘sja’”을 구성한다.

그 후 commit한다.





1. 강의 변경
2. 설명

강의 변경은 트랜잭션 T7에 해당한다.

CB\_1\_1, CB\_1\_2 과 TF\_1\_1, TF\_1\_2, TF\_1\_4 을 사용하며, CB 에서 칼럼을 선택할 수 있고, TF 에 각기 그에 맞는 키워드를 넣을 수 있다.

UI에서 TF\_1\_4 에 입력 받은 course\_id에 해당하는 레코드의 10개 칼럼 중 키워드로 1~2개를 선택하여 ( course\_id, title, semester, grade, classroom, time\_slot\_id, dept\_name, credit, tch\_name, remaining\_seat ) CB\_1 에서 고르고, TF\_1 에서 입력 받은 값으로 레코드의 선택한 칼럼들의 데이터를 입력한 데이터로 변경한다.

다만, primary key인 6개 칼럼( course\_id, title, semester, grade, tch\_name, remaining\_seat) 은 모든 CB\_1, TF\_1 에서 바꿀 수 있으나, non primary key인 나머지 4개 칼럼( classroom, titme\_slot\_id, dept\_name, credit ) 은 CB\_1\_1 과 TF\_1\_1 에서만 변경 가능하다.

변경이 완료되면 변경된 해당 레코드를 테이블에서 보여준다.

1. 구현 요건 충족

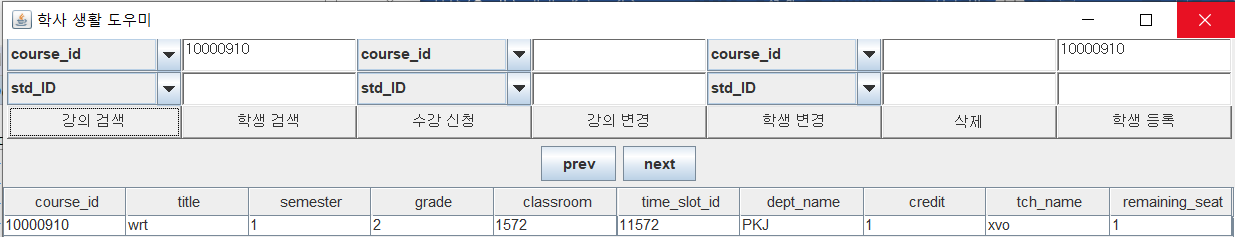
구현 요건 : 복수의 SQL 로 이뤄진 변경 트랜잭션

입력되는 키워드의 수에 따라, SQL문 S7을 실행하여, 외래 키가 설정된 course, takes, teaches 테이블 간의 데이터 UPDATE를 위한 각기 다른 TRIGGER를 생성한다.

그 후, update SQL문을 실행한다.

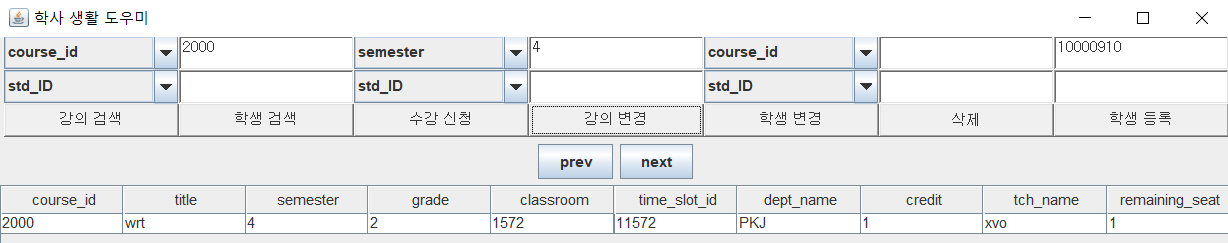
1. 실행화면
2. 변경 전

Course\_id = 10000910 인 레코드가 존재하고, semester = 1이다.

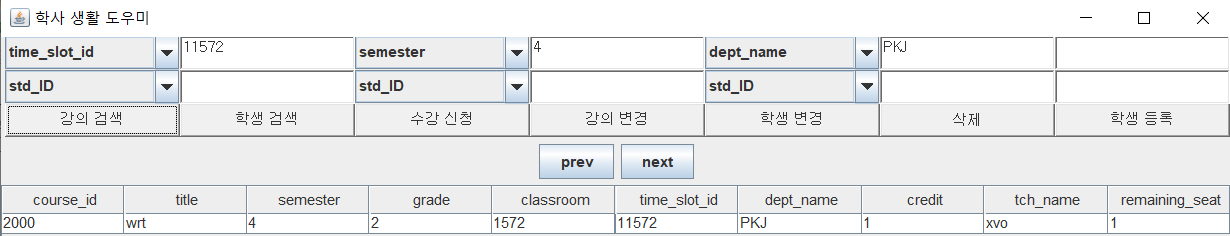


1. 변경 후

Course\_id = 10000910 -> 2000, semester = 1->4 변경한다.



1. 확인



1. 소스 코드

UI로부터 칼럼 index 와 키워드를 각각 inx, item 라는 매개변수로 받아온다.

매개변수로 받은 index와 키워드의 개수에 따라 동적 SQL문을 다르게 설정한다.

다시 말해, 키워드가 한 개이면 SQL문 S7-1을, 키워드가 두 개이면 SQL문 S7-2 으로 동적 SQL문을 설정한다.

여기서 CB\_1\_1 에서 선택한 칼럼이 course 테이블의 primary key인 지 아닌 지 검사하는 과정을 거친다.

Update를 하기 위해, 입력 받은 칼럼의 index 와 키워드인 get\_n 을 넣어 만든 update trigger를 정적 SQL로 먼저 실행한다.

그 후, update를 동적 SQL문으로 구성하여 진행한다.

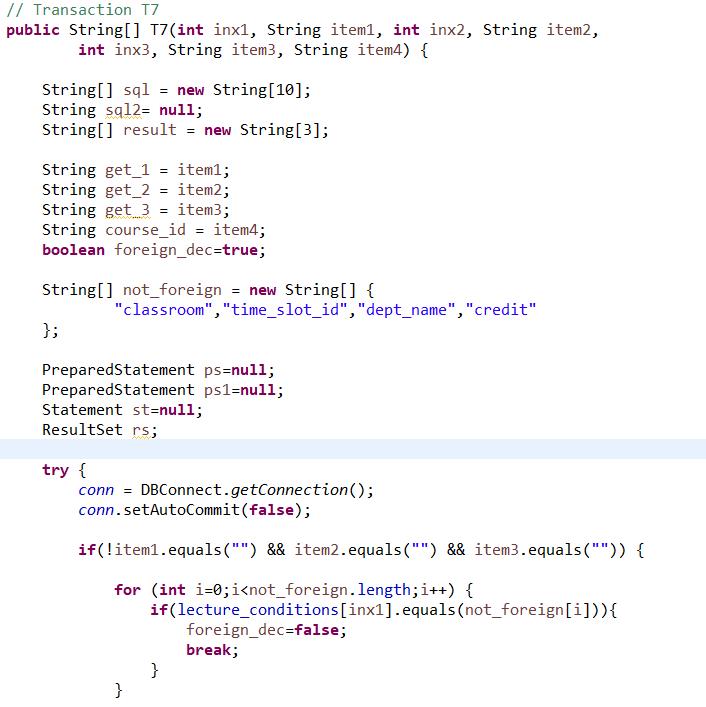
마지막으로, 변경 대상의 변경 결과를 테이블에서 보여준다.

그 후 commit한다.

가령, 위의 실행 예시에서, 정적 SQL문으로 UPDATE TRIGGE을 먼저 실행한다면,

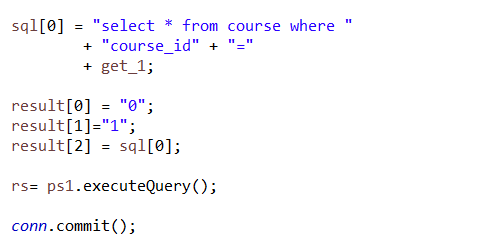
S7-2-1 : “create or replace trigger update\_trigger1 after update on course for each row begin update takes set course\_id = 2000, semester = 4 where course\_id = 10000910; end;”

S7-2-3 : “update course set course\_id = 2000, semester = 4 where course\_id = 10000910”









1. 학생 등록
2. 트랜잭션 설계

학생 등록은 트랜잭션 T6에 해당하며 student 테이블에 레코드를 삽입한다.

CB\_2 과 TF\_2 을 사용하며, CB\_2 에서 칼럼을 선택할 수 있고, TF\_2 에 각기 그에 맞는 키워드를 넣을 수 있다.

삽입 완료 후, 삽입된 레코드를 보여준다.

1. 구현 요건 충족

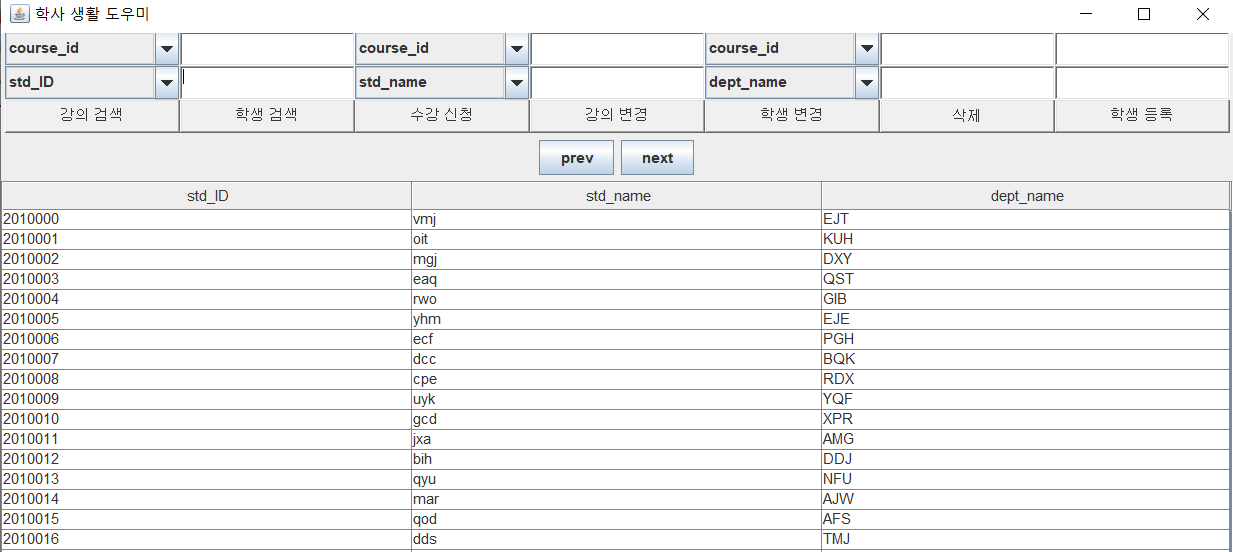
구현 요건 : 복수의 SQL 로 구성된 변경 트랜잭션.

선택된 3개의 칼럼과 각기 입력된 3개의 키워드로 동적 insert SQL 문 S8-2을 구성한다.

그 후, student 테이블의 자식 테이블인 takes 테이블에도 같은 std\_id과 임의의 값으로 구성된 동적 SQL문 S8-3 을 구성한다.

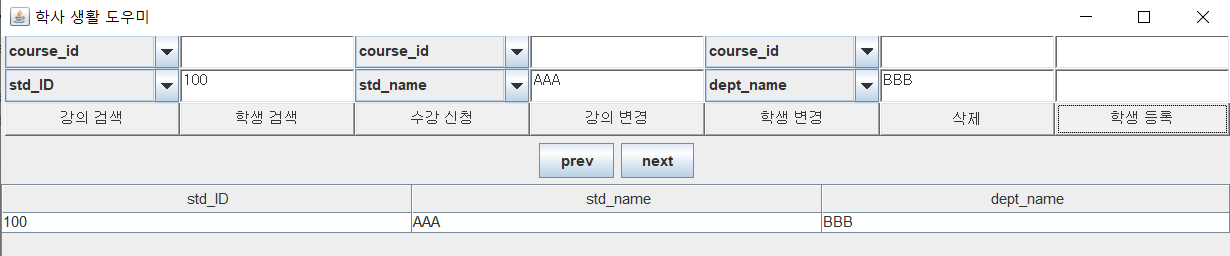
1. 실행화면
2. 삽입 전

Std\_id 를 기준으로 내림차순 정렬된 초기 테이블에서, Std\_id = 100 인 레코드는 존재하지 않는다.

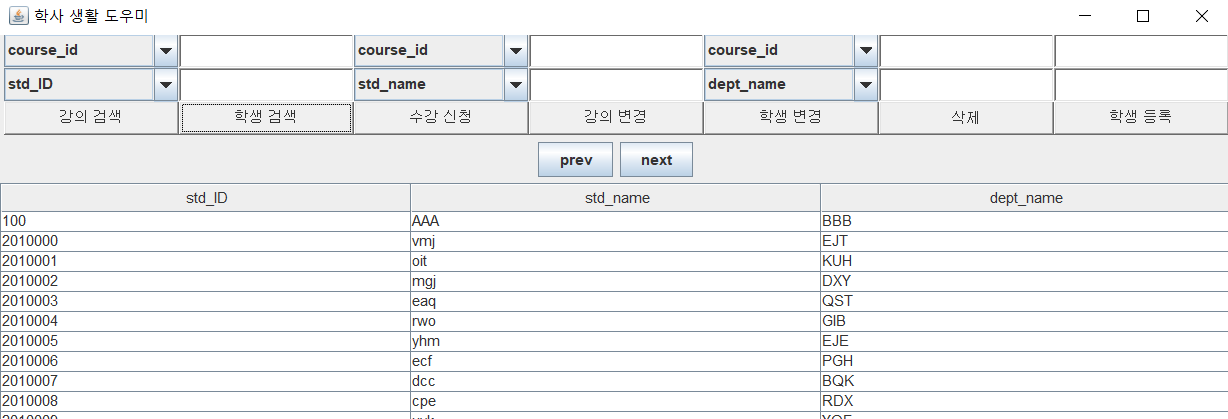


1. 삽입 후

Std\_ID = 100 , STD\_NAME = AAA, DEPT\_NAME = BBB 인 레코드를 삽입한다.



1. 확인



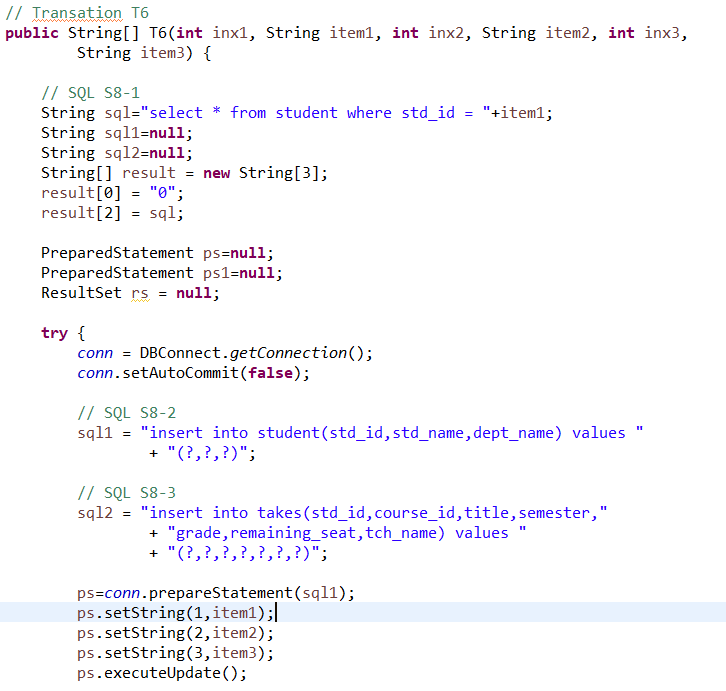
1. 소스 코드

UI로부터 키워드를 std\_id, std\_name, dept\_name 순서대로 item 라는 매개변수로 받아온다.

받은 item으로 동적 insert SQL문 S8-2을 구성하여 실행하여 삽입을 완료하고,

받은 STD\_ID로 동적 insert SQL문 S8-3을 구성하여 takes 테이블에 새로운 데이터를 삽입한다.

그 후 commit한다.





1. 학생 삭제
2. 트랜잭션 설계

학생 삭제는 트랜잭션 T4에 해당하며 student 테이블에서 레코드를 삭제한다.

TF\_2\_1에 삭제하려는 레코드의 std\_id를 삽입하여 해당 레코드를 삭제한다.

1. 구현 요건 충족

구현 요건 : 검색한 대상이 존재할 경우, 삭제 (변경) 진행하는 혼합 트랜잭션

TF\_2\_1 에서 얻은 STD\_ID로 해당 데이터가 테이블에 존재하는 지 검색하는 동적 SQL 문 S4-1 을 실행한다.

그 후, 해당 STD\_ID를 가진 레코드를 takes 테이블에서 삭제하는 동적 SQL문 S4-2와, student 테이블에서 삭제하는 S4-2를 실행한다.

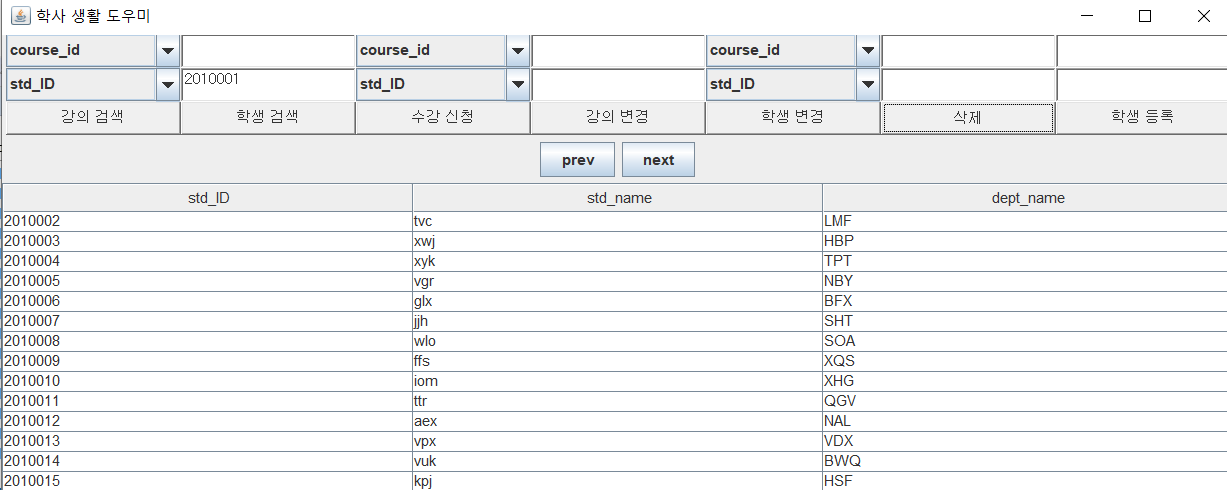
1. 실행화면
2. 삭제 전

Std\_id 를 기준으로 내림차순 정렬된 초기 테이블에서, Std\_id = 2010001인 레코드가 존재한다.



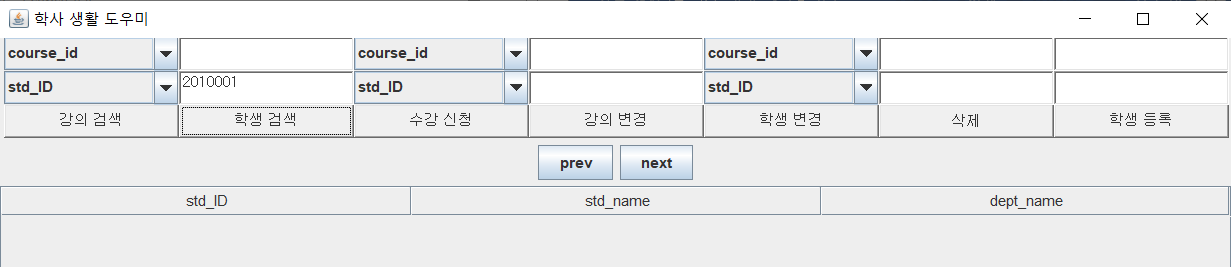
1. 삭제 후

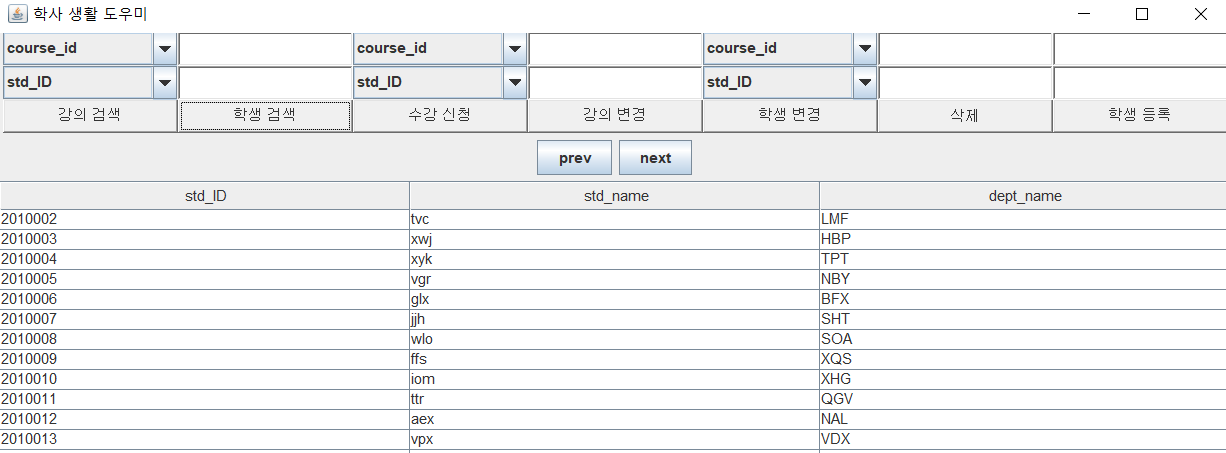
Std\_id = 2010001 인 레코드를 삭제한다.



1. 확인

Std\_id = 2010001 으로 학생 검색 결과, 해당 레코드는 삭제되었다.



Std\_id를 기준으로 내림차순으로 정렬된 student 테이블에서도, std\_id = 2010001 인 레코드는 더 이상 존재하지 않는다.  


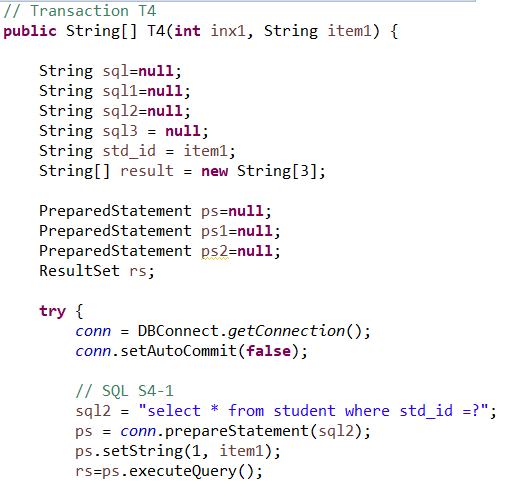
1. 소스 코드

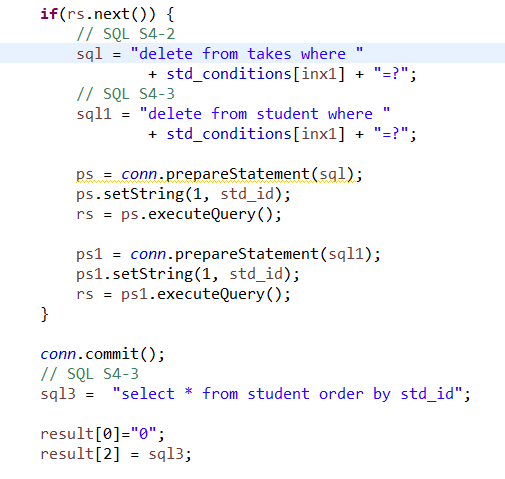
UI로부터 키워드를 std\_id, std\_name, dept\_name 순서대로 item 라는 매개변수로 받아온다.

받은 item으로 동적 insert SQL문 S8-2을 구성하여 실행하여 삽입을 완료하고,

받은 STD\_ID로 동적 insert SQL문 S8-3을 구성하여 takes 테이블에 새로운 데이터를 삽입한다.

그 후 commit한다.





1. 강의 삭제
2. 트랜잭션 설계

강의 삭제는 트랜잭션 T3에 해당하며 course 테이블에서 레코드를 삭제한다.

TF\_2\_1에 삭제하려는 레코드의 std\_id를 삽입하여 해당 레코드를 삭제한다.

이것은 특정 course\_id를 가진 레코드와 관련된 데이터를 모든 테이블에서 완전히 삭제한다. UI에서 TF\_1\_1에 입력 받은 course\_id를 가진 강의의 time\_slot\_id를 검색하여, course 테이블에서는 course\_id를 기반으로, time\_slot 테이블에서는 time\_slot\_id를 기반으로 완전히 데이터를 삭제한다.

1. 구현 요건 충족

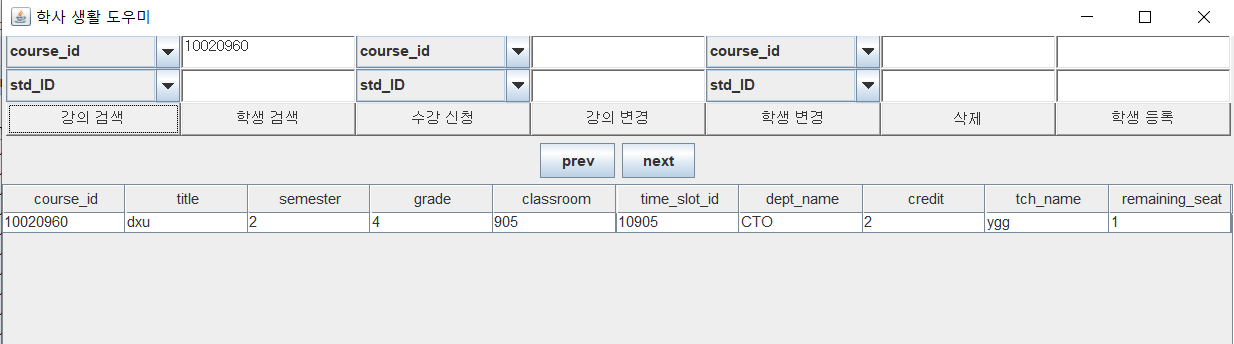
구현 요건 : BE 혼합 트랜잭션

TF\_1\_1 에서 얻은 course\_ID로 해당 데이터의 time\_slot\_id를 찾는 동적 SQL 문 S3-1을 구성한다.

그 후, 해당 course\_ID를 가진 레코드를 COURSE 테이블에서 삭제하는 동적 SQL문 S3-2와, time\_slot\_id 테이블에서 삭제하는 동적 SQL문 S3-3를 실행한다.

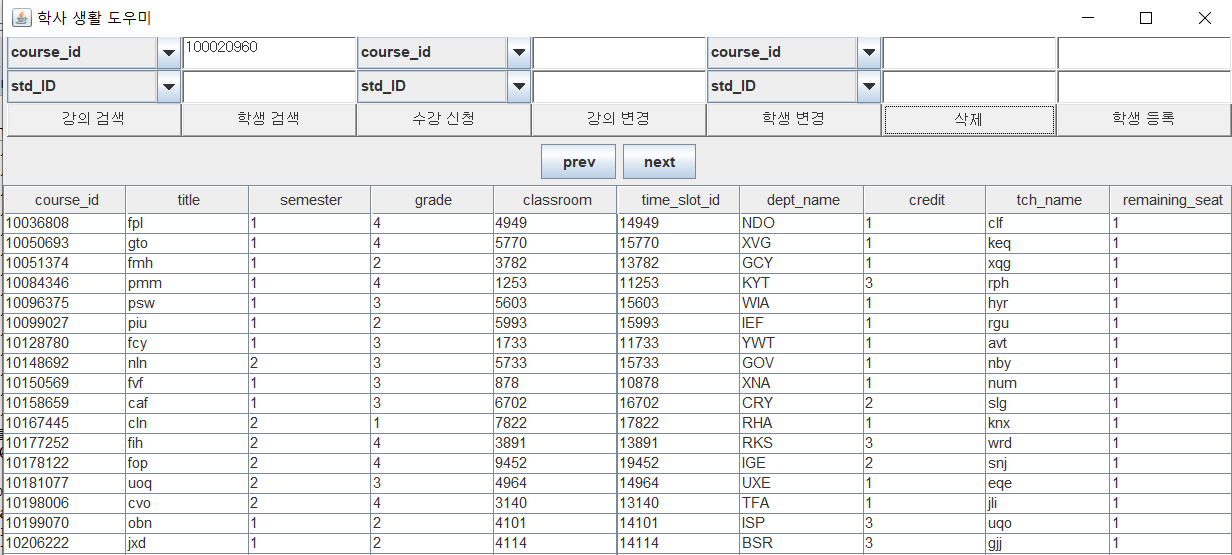
1. 실행화면
2. 삭제 전

Course\_id = 100020960 인 레코드가 course 테이블에 존재한다.



1. 삭제 후

Course\_id = 100020960인 레코드가 course\_id 기준으로 내림차순 정렬된 course 테이블에서 존재하지 않는다.



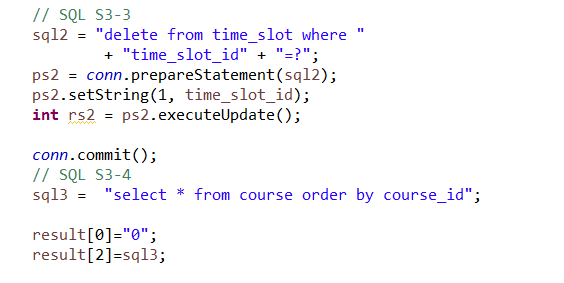
1. 소스 코드

TF\_1\_1 에서 얻은 course\_ID로 해당 데이터의 time\_slot\_id를 찾는 동적 SQL 문 S3-1을 구성한다.

그 후, 해당 course\_ID를 가진 레코드를 COURSE 테이블에서 삭제하는 동적 SQL문 S3-2와, time\_slot\_id 테이블에서 삭제하는 동적 SQL문 S3-3를 실행한다.

그 후 commit한다.





1. 학생 변경
2. 트랜잭션 설계

CB\_2 과 TF\_2 을 사용하며, CB 에서 칼럼을 선택할 수 있고, TF 에 각기 그에 맞는 키워드를 넣을 수 있다.

트랜잭션 T8에 해당하며, UI에서 TF\_2\_4입력 받은 std\_id로 해당 레코드를 검색하고, 그것의 3개 칼럼 중 키워드로 1~3개를 선택하여 (std\_id, std\_name, dept\_name ) UI에 입력 받은 값으로 레코드의 각 칼럼의 데이터를 변경한다. 다만, TF\_1에서는 primary key인 std\_id의 변경될 값을 입력 받고, 나머지 두 텍스트 필드에서 primary key가 아닌 값 2개 (std\_name, dept\_name)을 입력받는다.

1. 구현 요건 충족

구현 요건 : BE 혼합 트랜잭션

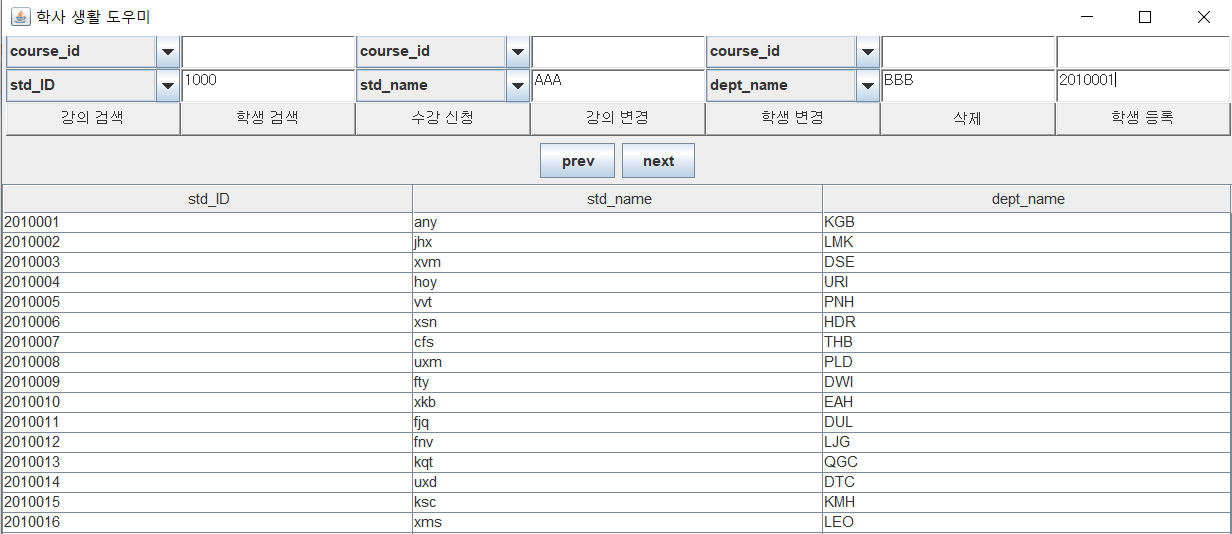
TF\_2\_4 에서 얻은 std\_ID로 해당 데이터가 존재하는 지, 그리고 해당 데이터의 std\_id 에 해당하는 칼럼의 index를 얻는 동적 SQL문 S8-1을 실행한다.

그 후, UI에서 입력 받은 키워드의 개수에 따라 각기 다른 UPDATE 동적 SQL문과, student 테이블의 자식 테이블인 takes 테이블에서도 같은 update를 진행하기 위한 정적 UPDATE TRIGGER SQL문을 구성한다.

그 후 UPDATE SQL문을 실행한다.

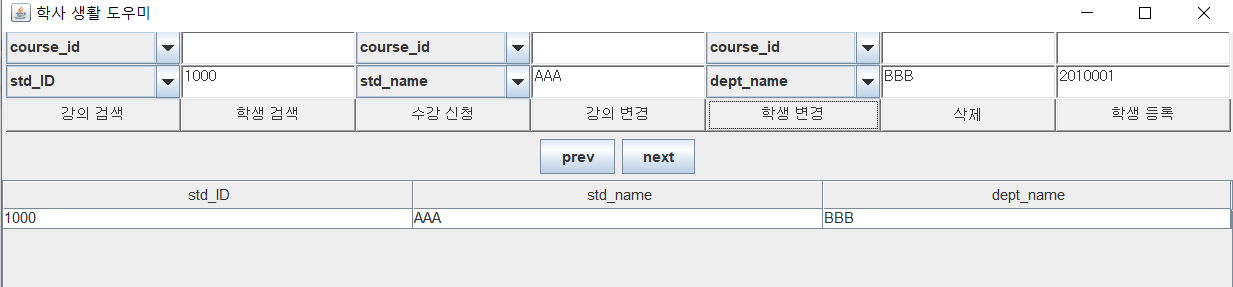
1. 실행화면
2. 변경 전

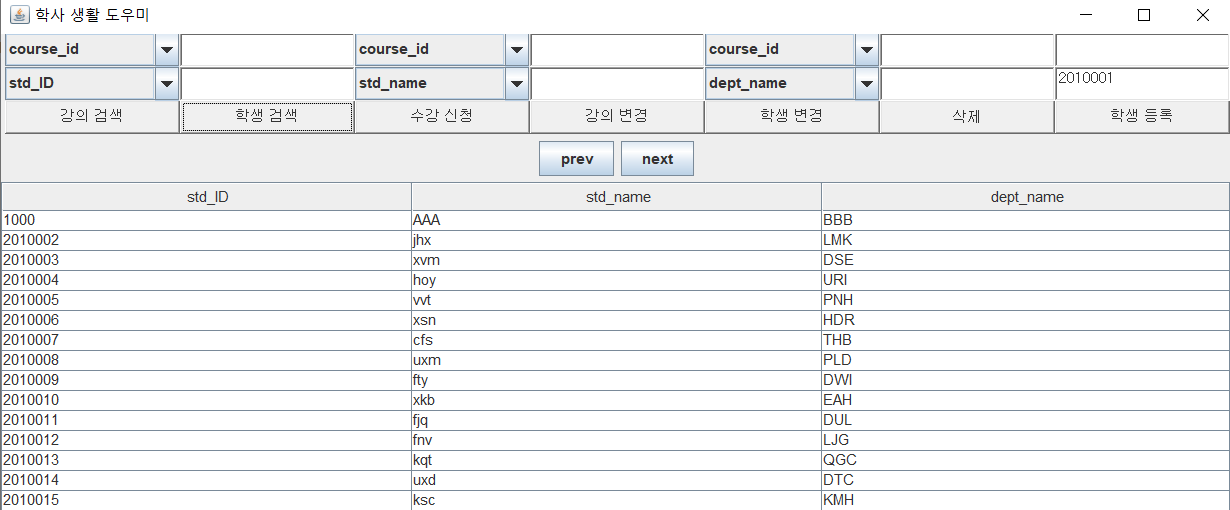
Std\_id = 2010001 인 레코드가 student 테이블에 존재한다.



1. 변경 후

Course\_id = 100020960 - > 1000, std\_name = any -> AAA, dept\_name = KGB -> BBB인 레코드로 변경된다.





1. 소스 코드

TF\_2\_4 에서 얻은 std\_ID로 해당 데이터가 존재하는 지, 그리고 해당 데이터의 std\_id 에 해당하는 칼럼의 index를 얻는 동적 SQL문 S8-1을 실행한다.

그 후, UI에서 입력 받은 키워드의 개수에 따라 각기 다른 UPDATE 동적 SQL문과, student 테이블의 자식 테이블인 takes 테이블에서도 같은 update를 진행하기 위한 정적 UPDATE TRIGGER SQL문을 구성한다.

그 후 UPDATE SQL문을 실행한다.

가령, 위의 실행 예시에서, STD\_ID = 2010001 을 대상으로 변경을 진행하는 과정은,

먼저 S8-1 로 해당 STD\_ID를 가진 레코드를 검색하고, 그 레코드의 std\_id 칼럼의 인덱스를 결과값으로 얻어, update trigger인 S8-4-1 = “create or replace trigger update\_trigger1 after update on student for each row begin update takes set std\_id = 1000, std\_name = AAA, dept\_name =BBB where std\_id = 2010001; end;”을 실행한다.

그 후, 동적 update SQL 인 S8-4-2 = “update student set std\_id = 1000, std\_name = AAA, dept\_name =BBB where std\_id = 2010001”을 실행한다.

그 후 commit한다.

