

# 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
4. 관계 DB 스키마 : "관계 DB 스키마.pdf" 로 첨부.
5. Create table 문 : "create\_table.rtf" (워드 패드) 로 첨부
6. Load한 레코드 수 : 5개 테이블에 각 1만 개

```
SQL> select count(*) from course;

COUNT(*)
-----
10000

SQL> select count(*) from student;

COUNT(*)
-----
10000

SQL> select count(*) from takes;

COUNT(*)
-----
10000

SQL> select count(*) from time_slot;

COUNT(*)
-----
10000

SQL> select count(*) from teaches;

COUNT(*)
-----
10000
```

## 7. 프로그램 GUI

학사 생활 도우미

course_id ▼		course_id ▼		course_id ▼		
std_ID ▼		std_ID ▼		std_ID ▼		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생 등록

prev    next

course_id	title	semester	grade	classroom	time_slot_id	dept_name	credit	tch_name	remaining_seat
10006794	vsk	1	4	985	10985	BLC	3	qpn	1
10018771	hml	2	3	503	10503	VLP	1	yos	1
10036800	gpq	1	2	5691	15691	GCV	2	cyc	1
10061593	dkf	1	3	1163	11163	VFQ	1	qam	1
10111614	gyh	1	2	3994	13994	ELO	3	nrm	1
10121866	tlr	1	2	2014	12014	FPS	2	vic	1
10128915	bgh	1	3	6040	16040	TYT	1	dyy	1
10152423	yhs	1	2	4621	14621	PTV	2	orj	1
10157846	hyt	2	1	7255	17255	YLH	3	sfi	1
10173853	vcx	1	1	2519	12519	YRK	2	ghg	1
10200798	xvg	2	1	1164	11164	ORA	2	jrk	1
10204576	tss	2	4	8176	18176	JKL	1	ibt	1
10210144	obx	1	4	2241	12241	PGB	3	ulw	1
10212608	sms	2	4	5009	15009	TTA	2	pnd	1
10222523	mfc	2	3	247	10247	VGR	2	dhr	1
10228739	icc	1	2	4572	14572	USD	1	rja	1
10230617	ffr	1	4	3428	13428	YNQ	3	hcq	1
10232969	esn	1	1	6543	16543	AUY	2	dmd	1
10238383	tut	1	2	581	10581	TDW	2	pur	1
10279830	ghb	2	1	6580	16580	XCD	3	ogv	1
10284165	rvk	2	1	6410	16410	EWV	2	lpk	1
10288820	vfh	1	2	7434	17434	DYL	1	kou	1
10296718	adh	1	2	8711	18711	EFK	1	byx	1
10309939	ufv	1	4	1465	11465	REV	1	jbf	1
10310793	kky	1	3	8348	18348	TOI	2	amr	1
10312817	cfv	2	2	4550	14550	LNY	1	uuu	1
10337280	gry	2	3	7741	17741	ARQ	3	boc	1
10348511	bka	2	3	6924	16924	JOR	3	wfy	1
10359446	hfg	1	2	2393	12393	YAR	1	pta	1

## 8. 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.

## 9. 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

## 10. 기능, 트랜잭션, 구현 요건 충족, 소스 코드 및 프로그램 화면 캡처 및 설명

### 1) 강의 검색

#### A. 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.

## B. 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.

## C. 실행화면

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

🖨️ 학사 생활 도우미

grade	▼ 2	credit	▼ 3	semester	▼ 1			
std_ID	▼	std_ID	▼	course_id	▲			
강의 검색		학생 검색		수강 신청		강의 변경		
						삭제	학생 등록	
						prev	next	
course_id	title	semester	grade	classroom	time_slot	credit	tch_name	remaining_seat
10006439	cql	1	1	7867	17867		jfi	1
10008098	exk	2	3	4539	14539		vcb	1
10045616	nhy	1	4	1636	11636		mih	1
10047883	psv	2	2	5555	15555	QSQ	ujn	1
10048718	swp	1	4	1532	11532	IGU	swa	1
10066517	oxw	2	3	1572	11572	GTK	qix	1
10072851	vcn	2	1	9124	19124	IWI	fdq	1
10079080	ric	2	3	7153	17153	YNR	nol	1
10091598	bup	2	3	2963	12963	FFG	cce	1
10105047	lcb	1	1	6825	16825	ONS	gwh	1
10105274	dmf	1	1	4284	14284	XEK	pji	1
10111823	tvu	2	1	2520	12520	EXQ	poc	1
10112767	fct	1	4	6772	16772	ULS	pnq	1
10115537	iet	2	4	4864	14864	GRX	hin	1
10122853	xeh	1	3	9191	19191	FOI	krj	1
10136548	vkf	2	3	3369	13369	VTY	yld	1
10138832	sps	1	3	8095	18095	GKY	khs	1
10141693	lkw	2	1	3999	13999	RFU	cer	1
10142809	jav	1	3	1865	11865	GJO	flv	1
10165648	llw	1	3	9277	19277	JEL	lfh	1
10172516	aus	2	1	462	10462	KOM	pkm	1
10180530	iun	2	2	6611	16611	UBR	tgs	1
10193679	okf	2	4	352	10352	NMN	oon	1
10196197	aji	1	3	7458	17458	FYY	ojr	1
10198300	wun	1	1	3925	13925	TRI	ovi	1

## D. 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, 키워드가 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.

```

// Transaction T1
public String[] T1(int inx1, String item1, int inx2, String item2, int inx3,
    String item3) {

    String sql=null;
    // SQL S1-1
    String sql1="select * from course";
    String[] result = new String[6];

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    ResultSet rs;

    int temp1 = item1.length();
    int temp2 = item2.length();
    int temp3 = item3.length();

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);

        ps=conn.prepareStatement(sql1);
        rs=ps.executeQuery();

        if (temp1 !=0 && temp2 ==0 && temp3 ==0) {
            // SQL S2-1
            sql = "select * from course where "
                + lecture_conditions[inx1] + "=?";

            result[0]="1";
            result[1]="1";
            result[3] = item1;
        }
    }
}

```

```

else if (temp1 !=0 && temp2 !=0 && temp3 ==0) {
    // SQL S2-2
    sql = "select * from course where "
        + lecture_conditions[inx1] + "=? "
        + " and " +
        lecture_conditions[inx2] + "=?";

    result[0]="1";
    result[1]="2";
    result[3] = item1;
    result[4] = item2;
}
else if(temp1 !=0 && temp2 !=0 && temp3 !=0) {
    // SQL S2-3
    sql = "select * from course where "
        + lecture_conditions[inx1] + "=? "
        + " and " +
        lecture_conditions[inx2] + "=? "
        + " and " +
        lecture_conditions[inx3] + "=?";

    result[0]="1";
    result[1]="3";
    result[3] = item1;
    result[4] = item2;
    result[5] = item3;
}
else {
    // SQL S2-4
    sql = "select * from course order by course_id";
    result[0]="0";
}

result[2] = sql;

ps1=conn.prepareStatement(sql);
rs=ps.executeQuery();
conn.commit();

```

## 2) 학생 검색

### A. 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.

### B. 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.

### C. 실행화면

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

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

📖 학사 생활 도우미

course_id		course_id		course_id		
std_name		std_ID		std_ID		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
		prev	next			
std_ID	std_name			dept_name		
2010000	dgu			BLJ		
2010001	pqo			ROI		
2010002	jna			SYD		
2010003	wmy			CMB		
2010004	vfm			NUI		
2010005	tvr			LUQ		
2010006	fbj			CBJ		
2010007	sja			KFE		
2010008	evd			LRO		
2010009	jot			MGS		
2010010	hlt			HTO		
2010011	ahg			SKV		
2010012	pyo			JRG		
2010013	dxj			CGU		
2010014	mkb			UBW		
2010015	rfm			TXY		
2010016	ucu			YUL		
2010017	pgf			AXW		
2010018	awy			GGA		
2010019	omt			DYX		
2010020	hym			GOJ		
2010021	hlu			OUD		
2010022	rwd			WON		
2010023	qfr			FIW		
2010024	poi			TNF		
2010025	vhe			DRB		
2010026	aox			LBR		

📖 학사 생활 도우미

course_id		course_id		course_id		
std_name	sja	std_ID		std_ID		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
		prev	next			
std_ID	std_name			dept_name		
2010007	sja			KFE		

### D. 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, 키워드가 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.

```
// Transaction T2
public String[] T2(int inx1, String item1, int inx2, String item2, int inx3,
    String item3) {

    String sql=null;
    // SQL S1-1
    String sql1="select * from student";
    String[] result = new String[6];

    int temp1 = item1.length();
    int temp2 = item2.length();
    int temp3 = item3.length();

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    ResultSet rs;

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);

        ps=conn.prepareStatement(sql1);
        rs=ps.executeQuery();

        if (temp1 !=0 && temp2 ==0 && temp3 ==0) {
            // SQL S2-1
            sql = "select * from student where "
                + std_conditions[inx1] + "=?";

            result[0]="1";
            result[1]="1";
            result[3] = item1;
        }
    }
```



```

else if (temp1 !=0 && temp2 !=0 && temp3 ==0) {
    // SQL S2-2
    sql = "select * from student where "
        + std_conditions[inx1] + "=? "
        + " and " +
        std_conditions[inx2] + "=?";

    result[0]="1";
    result[1]="2";
    result[3] = item1;
    result[4] = item2;
}
else if(temp1 !=0 && temp2 !=0 && temp3 !=0) {
    // SQL S2-3
    sql = "select * from student where "
        + std_conditions[inx1] + "=? "
        + " and " +
        std_conditions[inx2] + "=? "
        + " and " +
        std_conditions[inx3] + "=?";

    result[0]="1";
    result[1]="3";
    result[3] = item1;
    result[4] = item2;
    result[5] = item3;
}
else {
    // SQL S2-4
    sql = "select * from student order by std_ID";
    result[0]="0";
}

result[2] = sql;

ps1=conn.prepareStatement(sql);
rs=ps.executeQuery();
conn.commit();

```

### 3) 강의 변경

#### A. 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, time\_slot\_id, dept\_name, credit ) 은 CB\_1\_1 과 TF\_1\_1 에서만 변경 가능하다.

변경이 완료되면 변경된 해당 레코드를 테이블에서 보여준다.

## B. 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.

## C. 실행화면

### a. 변경 전

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

학사 생활 도우미

course_id	▼	10000910	course_id	▼		course_id	▼		10000910
std_ID	▼		std_ID	▼		std_ID	▼		
강의 검색		학생 검색		수강 신청		강의 변경		학생 변경	
								삭제	
prev next									
course_id		title		semester		grade		classroom	
10000910		wrt		1		2		1572	
								time_slot_id	
								11572	
								dept_name	
								PKJ	
								credit	
								1	
								tch_name	
								xvo	
								ren	
								1	

### b. 변경 후

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

학사 생활 도우미

course_id	▼	2000	semester	▼	4	course_id	▼		10000910
std_ID	▼		std_ID	▼		std_ID	▼		
강의 검색		학생 검색		수강 신청		강의 변경		학생 변경	
								삭제	
prev next									
course_id		title		semester		grade		classroom	
2000		wrt		4		2		1572	
								time_slot_id	
								11572	
								dept_name	
								PKJ	
								credit	
								1	
								tch_name	
								xvo	
								ren	
								1	

### c. 확인

학사 생활 도우미

time_slot_id	▼	11572	semester	▼	4	dept_name	▼	PKJ	
std_ID	▼		std_ID	▼		std_ID	▼		
강의 검색		학생 검색		수강 신청		강의 변경		학생 변경	
								삭제	
prev next									
course_id		title		semester		grade		classroom	
2000		wrt		4		2		1572	
								time_slot_id	
								11572	
								dept_name	
								PKJ	
								credit	
								1	
								tch_name	
								xvo	

## D. 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 TRIGGER를 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"

```
// Transaction T7
public String[] T7(int inx1, String item1, int inx2, String item2,
    int inx3, String item3, String item4) {

    String[] sql = new String[10];
    String sql2= null;
    String[] result = new String[3];

    String get_1 = item1;
    String get_2 = item2;
    String get_3 = item3;
    String course_id = item4;
    boolean foreign_dec=true;

    String[] not_foreign = new String[] {
        "classroom", "time_slot_id", "dept_name", "credit"
    };

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    Statement st=null;
    ResultSet rs;

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);

        if(!item1.equals("") && item2.equals("") && item3.equals("")) {

            for (int i=0;i<not_foreign.length;i++) {
                if(lecture_conditions[inx1].equals(not_foreign[i])){
                    foreign_dec=false;
                    break;
                }
            }
        }
    }
```

```

if(foreign_dec) {

    // SQL S7-1-1
    sql[1] = "create or replace trigger update_trigger1 "
        + "after update on course "
        + "for each row "
        + "begin "
        + "update takes "
        + "set "+ lecture_conditions[inx1]
        + "="+ get_1
        + " where course_id=" +course_id+";"+" end;" ;

    // SQL S7-1-2
    sql[2] = "create or replace trigger update_trigger2 "
        + "after update on course "
        + "for each row "
        + "begin "
        + "update teaches "
        + "set "+ lecture_conditions[inx1]
        + "="+ get_1
        + " where course_id=" +course_id+";"+" end;" ;

    st=conn.createStatement();
    st.execute(sql[1]);
    st=conn.createStatement();
    st.execute(sql[2]);
}

// SQL S7-1-3
sql[9] = "update course set "+ lecture_conditions[inx1] + "="+"
    + " where course_id=" +course_id;

ps1 = conn.prepareStatement(sql[9]);
ps1.setString(1, get_1);
}

```

```

else if(!item1.equals("") && !item2.equals("") && item3.equals(""))
{
    // SQL S7-2-1
    sql[1] = "create or replace trigger update_trigger3 "
        + "after update on course "
        + "for each row "
        + "begin "
        + "update takes "
        + "set "+ lecture_conditions[inx1]
        + "=" + get_1 + ","
        + lecture_conditions[inx2]
        + "=" + get_2
        + " where course_id=" + course_id + ";" + " end;" ;

    // SQL S7-2-2
    sql[2] = "create or replace trigger update_trigger4 "
        + "after update on course "
        + "for each row "
        + "begin "
        + "update teaches "
        + "set "+ lecture_conditions[inx1]
        + "=" + get_1 + ","
        + lecture_conditions[inx2]
        + "=" + get_2
        + " where course_id=" + course_id + ";" + " end;" ;

    // SQL S7-2-3
    sql[9] = "update course set "+ lecture_conditions[inx1] + "=?,"
        + lecture_conditions[inx2] + "=?"
        + " where course_id=" + course_id;

    st=conn.createStatement();
    st.execute(sql[1]);
    st=conn.createStatement();
    st.execute(sql[2]);

    ps1 = conn.prepareStatement(sql[9]);
    ps1.setString(1, get_1);
    ps1.setString(2, get_2);
}

sql[0] = "select * from course where "
    + "course_id" + "="
    + get_1;

result[0] = "0";
result[1]="1";
result[2] = sql[0];

rs= ps1.executeQuery();

conn.commit();

```

#### 4) 학생 등록

##### A. 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.

## B. 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\_id와 임의의 값으로 구성된 동적 SQL문 S8-3 을 구성한다.

## C. 실행화면

### a. 삽입 전

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

학사 생활 도우미						
course_id		course_id		course_id		
std_ID		std_name		dept_name		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
prev			next			
std_ID	std_name			dept_name		
2010000	vmj			EJT		
2010001	oit			KUH		
2010002	mgj			DXY		
2010003	eaq			QST		
2010004	rwo			GIB		
2010005	yhm			EJE		
2010006	ecf			PGH		
2010007	dcc			BQK		
2010008	cpe			RDX		
2010009	uyk			YQF		
2010010	gcd			XPR		
2010011	jxa			AMG		
2010012	bih			DDJ		
2010013	qyu			NFU		
2010014	mar			AJW		
2010015	qod			AFS		
2010016	dds			TMJ		

### b. 삽입 후

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

학사 생활 도우미						
course_id		course_id		course_id		
std_ID	100	std_name	AAA	dept_name	BBB	
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생 등록
prev			next			
std_ID	std_name			dept_name		
100	AAA			BBB		

### c. 확인

course_id		course_id		course_id		
std_ID		std_name		dept_name		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
			prev	next		
std_ID		std_name		dept_name		
100		AAA		BBB		
2010000		vmj		EJT		
2010001		oit		KUH		
2010002		mgj		DXY		
2010003		eaq		QST		
2010004		rwo		GIB		
2010005		yhm		EJE		
2010006		ecf		PGH		
2010007		dcc		BQK		
2010008		cpe		RDX		
2010009		...		...		

#### D. 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.

```
// Transation T6
public String[] T6(int inx1, String item1, int inx2, String item2, int inx3,
    String item3) {

    // SQL S8-1
    String sql="select * from student where std_id = "+item1;
    String sql1=null;
    String sql2=null;
    String[] result = new String[3];
    result[0] = "0";
    result[2] = sql;

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    ResultSet rs = null;

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);

        // SQL S8-2
        sql1 = "insert into student(std_id,std_name,dept_name) values "
            + "(?,?,?)";

        // SQL S8-3
        sql2 = "insert into takes(std_id,course_id,title,semester,"
            + "grade,remaining_seat,tch_name) values "
            + "(?,?,?,?,?,?,?)";

        ps=conn.prepareStatement(sql1);
        ps.setString(1,item1);
        ps.setString(2,item2);
        ps.setString(3,item3);
        ps.executeUpdate();
```

```

ps1=conn.prepareStatement(sql2);
ps1.setString(1,item1);
ps1.setString(2,"1000000");
ps1.setString(3,"aaa");
ps1.setString(4,"2");
ps1.setString(5,"2");
ps1.setString(6,"2");
ps1.setString(7,"bbb");
ps1.executeUpdate();

conn.commit();

}catch (Exception e) {
    e.printStackTrace();
} finally {
    if (ps != null) {
        try {
            ps.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (ps1 != null) {
        try {
            ps1.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}
return result;

```

## 5) 학생 삭제

### A. 트랜잭션 설계

학생 삭제는 트랜잭션 T4에 해당하며 student 테이블에서 레코드를 삭제한다.

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

### B. 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.

### C. 실행화면



#### d. 삭제 전

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

학사 생활 도우미						
course_id		course_id		course_id		
std_ID		std_ID		std_ID		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
			prev	next		
std_ID	std_name			dept_name		
2010001	cev			IMA		
2010002	tvc			LMF		
2010003	xwj			HBP		
2010004	xyk			TPT		
2010005	vgr			NBY		
2010006	glx			BFX		
2010007	jih			SHT		
2010008	wlo			SOA		
2010009	ffs			XQS		
2010010	iom			XHG		
2010011	ttr			QGV		
2010012	aex			NAL		
2010013	vpx			VDX		

#### e. 삭제 후

Std\_id = 2010001 인 레코드를 삭제한다.

학사 생활 도우미						
course_id		course_id		course_id		
std_ID	2010001	std_ID		std_ID		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
			prev	next		
std_ID	std_name			dept_name		
2010002	tvc			LMF		
2010003	xwj			HBP		
2010004	xyk			TPT		
2010005	vgr			NBY		
2010006	glx			BFX		
2010007	jih			SHT		
2010008	wlo			SOA		
2010009	ffs			XQS		
2010010	iom			XHG		
2010011	ttr			QGV		
2010012	aex			NAL		
2010013	vpx			VDX		
2010014	vuk			BWQ		
2010015	kpj			HSF		

#### f. 확인

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

학사 생활 도우미						
course_id		course_id		course_id		
std_ID	2010001	std_ID		std_ID		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생 등록
			prev	next		
std_ID	std_name			dept_name		

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

course_id		course_id		course_id		
std_ID		std_ID		std_ID		
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생
<div>prevnext</div>						
std_ID		std_name			dept_name	
2010002		tvc			LMF	
2010003		xwj			HBP	
2010004		xyk			TPT	
2010005		vgr			NBY	
2010006		glx			BFX	
2010007		jjh			SHT	
2010008		wlo			SOA	
2010009		ffs			XQS	
2010010		iom			XHG	
2010011		ttr			QGV	
2010012		aex			NAL	
2010013		vpx			VDX	

#### D. 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.

```
// Transaction T4
public String[] T4(int inx1, String item1) {

    String sql=null;
    String sql1=null;
    String sql2=null;
    String sql3 = null;
    String std_id = item1;
    String[] result = new String[3];

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    PreparedStatement ps2=null;
    ResultSet rs;

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);

        // SQL S4-1
        sql2 = "select * from student where std_id =?";
        ps = conn.prepareStatement(sql2);
        ps.setString(1, item1);
        rs=ps.executeQuery();
```

```

if(rs.next()) {
    // SQL S4-2
    sql = "delete from takes where "
        + std_conditions[inx1] + "=?";
    // SQL S4-3
    sql1 = "delete from student where "
        + std_conditions[inx1] + "=?";

    ps = conn.prepareStatement(sql);
    ps.setString(1, std_id);
    rs = ps.executeQuery();

    ps1 = conn.prepareStatement(sql1);
    ps1.setString(1, std_id);
    rs = ps1.executeQuery();
}

conn.commit();
// SQL S4-3
sql3 = "select * from student order by std_id";

result[0]="0";
result[2] = sql3;

```

## 6) 강의 삭제

### A. 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.

### B. 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.

### C. 실행화면

#### g. 삭제 전

Course\_id = 100020960 인 레코드가 course 테이블에 존재한다.



```

// Transaction T3
public String[] T3(int inx1, String item1) {

    String sql=null;
    String sql1=null;
    String sql2=null;
    String sql3 = null;
    String time_slot_id = null;
    String course_id = item1;
    String[] result = new String[3];

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    PreparedStatement ps2=null;
    ResultSet rs;

    // SQL S3-1
    sql = "select time_slot_id from course where "
        + lecture_conditions[inx1] + "="
        + ""+course_id+"";

    // SQL S3-2
    sql1 = "delete from course where "
        + lecture_conditions[inx1] + "=?";

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);
        ps = conn.prepareStatement(sql);
        rs = ps.executeQuery();
        while(rs.next()) {
            time_slot_id = rs.getString(1);
        }

        ps1 = conn.prepareStatement(sql1);
        ps1.setString(1, course_id);
        int rs1 = ps1.executeUpdate();

        // SQL S3-3
        sql2 = "delete from time_slot where "
            + "time_slot_id" + "=?";
        ps2 = conn.prepareStatement(sql2);
        ps2.setString(1, time_slot_id);
        int rs2 = ps2.executeUpdate();

        conn.commit();
        // SQL S3-4
        sql3 = "select * from course order by course_id";

        result[0]="0";
        result[2]=sql3;
    }
}

```

## 7) 학생 변경

### A. 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.

## B. 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.

## C. 실행화면

### i. 변경 전

Std\_id = 2010001 인 레코드가 student 테이블에 존재한다.

학사 생활 도우미

course_id		course_id		course_id		
std_ID	1000	std_name	AAA	dept_name	BBB	2010001
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생

prev next

std_ID	std_name	dept_name
2010001	any	KGB
2010002	jhx	LMK
2010003	xvm	DSE
2010004	hoy	URI
2010005	vvt	PNH
2010006	xsn	HDR
2010007	cfs	THB
2010008	uxm	PLD
2010009	fty	DWI
2010010	xkb	EAH
2010011	fjq	DUL
2010012	fnv	LJG
2010013	kqt	QGC
2010014	uxd	DTC
2010015	ksc	KMH
2010016	xms	LEO

### j. 변경 후

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

학사 생활 도우미

course_id		course_id		course_id		
std_ID	1000	std_name	AAA	dept_name	BBB	2010001
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생 등록
			prev		next	
std_ID		std_name		dept_name		
1000		AAA		BBB		

학사 생활 도우미

course_id		course_id		course_id		
std_ID		std_name		dept_name		2010001
강의 검색	학생 검색	수강 신청	강의 변경	학생 변경	삭제	학생 등록
			prev		next	
std_ID		std_name		dept_name		
1000		AAA		BBB		
2010002		jhx		LMK		
2010003		xvm		DSE		
2010004		hoy		URI		
2010005		vvt		PNH		
2010006		xsn		HDR		
2010007		cfs		THB		
2010008		uxm		PLD		
2010009		fty		DWI		
2010010		xkb		EAH		
2010011		fjq		DUL		
2010012		fnv		LJG		
2010013		kqt		QGC		
2010014		uxd		DTC		
2010015		ksc		KMH		

#### D. 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.

```

// Transaction T8
public String[] T8(int inx1, String item1, int inx2, String item2,
    int inx3, String item3, String item4) {

    String[] sql = new String[10];

    String get_1 = item1;
    String get_2 = item2;
    String get_3 = item3;
    String std_id = item4;
    String[] result = new String[3];
    result[0]="0";

    PreparedStatement ps=null;
    PreparedStatement ps1=null;
    Statement st=null;
    ResultSet rs;

    // SQL S8-1
    sql[8] = "select * from student where "
        + "std_id" + "="
        + get_1;

    result[2]=sql[8];

    try {
        conn = DBConnect.getConnection();
        conn.setAutoCommit(false);

        // SQL S8-1
        sql[0] = "select * from student where "
            + "std_id" + "=?";

        if(!item1.equals("") && item2.equals("") && item3.equals("")) {

            // SQL S8-2-1
            sql[1] = "create or replace trigger update_trigger1 "
                + "after update on student "
                + "for each row "
                + "begin "
                + "update takes "
                + "set "+ std_conditions[inx1]
                + "=" + get_1
                + " where std_id=" +std_id+";"+ " end;" ;

            st=conn.createStatement();
            st.execute(sql[1]);

            ps = conn.prepareStatement(sql[0]);
            ps.setString(1, get_1);
            rs=ps.executeQuery();
            int target_index = rs.findColumn("std_id");

            // SQL S8-2-2
            sql[9] = "update student set "+ std_conditions[target_index-1] + "=?"
                + " where std_id=" +std_id;

            ps1 = conn.prepareStatement(sql[9]);
            ps1.setString(1, get_1);
        }
    }

```



```

else if(!item1.equals("") && !item2.equals("") && item3.equals(""))
{
    // SQL S8-3-1
    sql[1] = "create or replace trigger update_trigger1 "
        + "after update on student "
        + "for each row "
        + "begin "
        + "update takes "
        + "set "+ std_conditions[inx1]
        + "=" + get_1
        + " where std_id=" +std_id+";"+" end;" ;

    st=conn.createStatement();
    st.execute(sql[1]);

    ps = conn.prepareStatement(sql[0]);
    ps.setString(1, get_1);
    rs=ps.executeQuery();
    int target_index = rs.findColumn("std_id");

    // SQL S8-3-2
    sql[9] = "update student set "+ std_conditions[target_index-1] + "=?,"
        + std_conditions[inx2] + "=?"
        + " where std_id=" +std_id;

    ps1 = conn.prepareStatement(sql[9]);
    ps1.setString(1, get_1);
    ps1.setString(2, get_2);
}

```

```

else if(!item1.equals("") && !item2.equals("") && !item3.equals(""))
{
    // SQL S8-4-1
    sql[1] = "create or replace trigger update_trigger1 "
        + "after update on student "
        + "for each row "
        + "begin "
        + "update takes "
        + "set "+ std_conditions[inx1]
        + "="+ get_1
        + " where std_id=" +std_id+";"+" end;" ;

    st=conn.createStatement();
    st.execute(sql[1]);

    ps = conn.prepareStatement(sql[0]);
    ps.setString(1, get_1);
    rs=ps.executeQuery();
    int target_index = rs.findColumn("std_id");

    // SQL S8-4-2
    sql[9] = "update student set "+ std_conditions[target_index-1] + "=?,"
        + std_conditions[inx2] + "=?,"
        + std_conditions[inx3] + "=?"
        + " where std_id=" +std_id;

    ps1 = conn.prepareStatement(sql[9]);
    ps1.setString(1, get_1);
    ps1.setString(2, get_2);
    ps1.setString(3, get_3);
}

rs= ps1.executeQuery();

conn.commit();

```