

제 4 장 SQL

- SQL 개요
- SQL 문법
- 오라클에서의 SQL 실습

질의어와 SQL

- ▶ Structured Query Language
- ▶ 1974년 IBM의 System R project에서 개발된 Sequel이란 언어에 기초
- ▶ 표준 질의어로 채택되어 널리 쓰이는 관계형 질의언어
 - ▶ 1986년 ANSI와 ISO에서 표준 질의어로 채택
 - ▶ 1992년 SQL2(SQL-92) 발표
 - ▶ 2003년 SQL3발표 (최신)
- ▶ 관계 대수나 관계 해석은 확실한 이론적 배경을 제공하나 상용으로 쓰이기에는 어렵고 적절치 않음
 - ▶ SQL은 자연어와 유사하고 비절차적 언어이므로 사용하기 용이함

SQL의 구성: DDL & DML

- ▶ SQL은 크게 DDL과 DML로 구성됨
- ▶ 데이터 정의 언어 (DDL: Data Definition Language)
 - ▶ 데이터 저장 구조를 명시하는 언어
 - ▶ 테이블 스키마의 정의, 수정, 삭제
- ▶ 데이터 조작 언어 (DML: Data Manipulation Language)
 - ▶ 사용자가 데이터를 접근하고 조작할 수 있게 하는 언어
 - ▶ 레코드의 검색(search), 삽입(insert), 삭제(delete), 수정(update)

데이터 정의 언어

- ▶ 테이블 생성 (create table)
- ▶ 기본키, 외래키 설정
- ▶ 테이블 삭제 (drop table)
- ▶ 테이블 수정 (alter table)

데이터 정의 언어

▶ 종류

- ▶ 테이블 생성
- ▶ 테이블 삭제
- ▶ 테이블 수정

▶ 필드의 Data type 종류

분류	표준 SQL	오라클	설명
문자	char(n)	char(n)	길이가 n byte인 고정길이 문자열 오라클의 경우 최대 2000byte까지 지정 가능
	varchar(n)	varchar2(n)	최대 길이가 n byte인 가변길이 문자열 오라클의 경우 최대 4000byte까지 지정 가능
숫자	int	int	정수형
	float	float	부동 소수
날짜 시간	date	date	년, 월, 일을 갖는 날짜형 오라클의 경우 날짜의 기본 형식은 'yy/mm/dd'이다.
	time timestamp	timestamp	년, 월, 일, 시, 분, 초를 갖는 날짜시간형

테이블 생성

▶ 형식

```
create table <테이블이름> (<필드리스트>)
```

- ▶ <필드리스트>는 '필드명 데이터타입'

▶ department 테이블을 생성하는 SQL문

(질의 1)

```
create table department
(
    dept_id          varchar2(10)    not null,
    dept_name        varchar2(14)    not null,
    office            varchar2(10)
)
```

- ▶ 키워드 **not null**은 해당 필드에 널을 허용하지 않는다는 것을 의미함

기본키, 외래키 설정

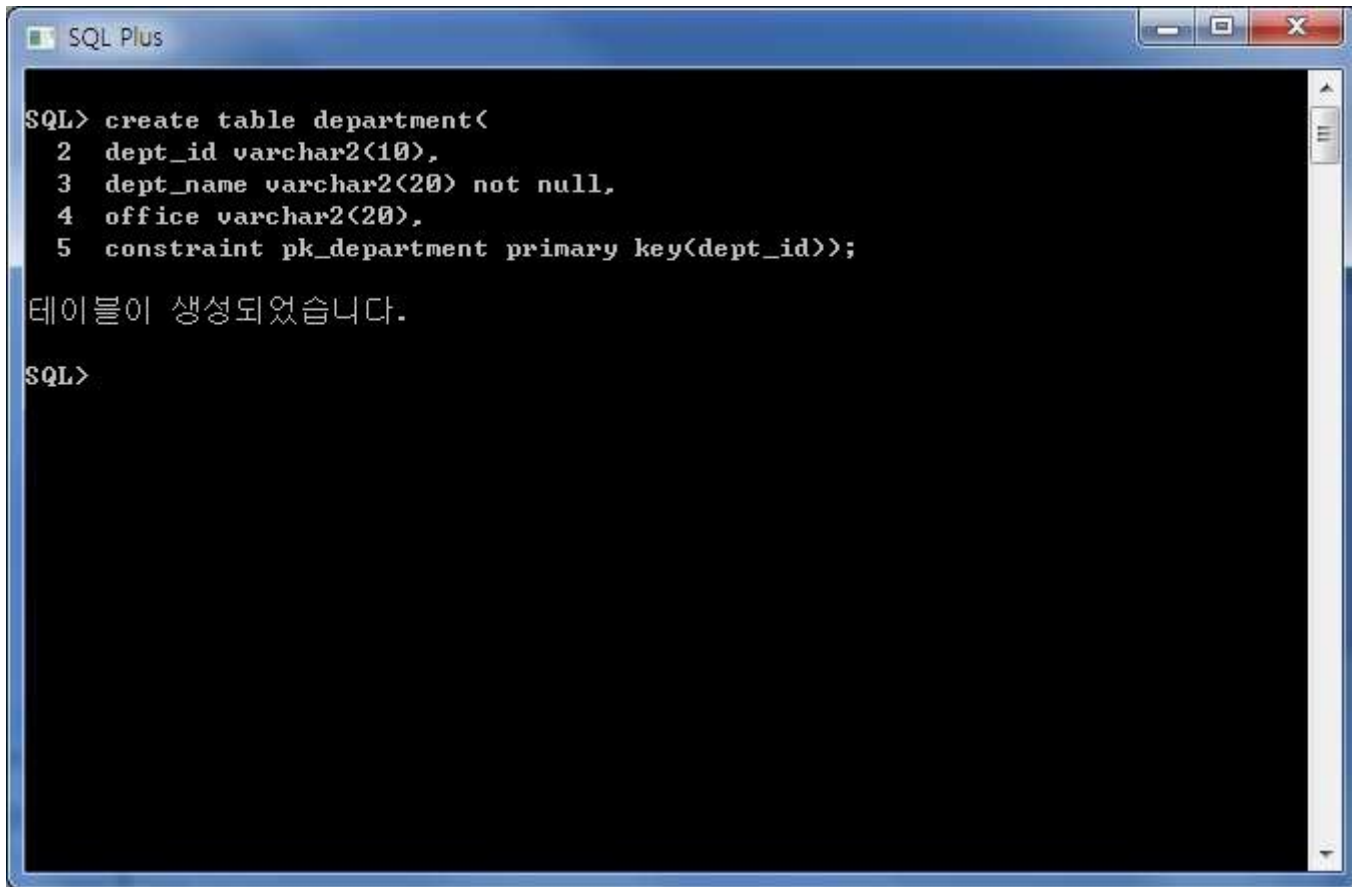
- ▶ 테이블을 생성할 기본키 역할을 하는 필드를 지정

(질의 2)

```
create table department
(
    dept_id          varchar2(10),
    dept_name        varchar2(20) not null,
    office           varchar2(20),
    constraint pk_department primary key(dept_id)
)
```

- ▶ pk_department: 제약식의 이름

기본키, 외래키 설정



```
SQL> create table department(  
  2  dept_id varchar2(10),  
  3  dept_name varchar2(20) not null,  
  4  office varchar2(20),  
  5  constraint pk_department primary key(dept_id));  
  
테이블이 생성되었습니다.  
  
SQL>
```


테이블 생성(student table)

- ▶ not null과 기본키를 지정한 student 테이블 생성 예

(질의 3)

```
create table student
(
    stu_id                varchar2(10),
    resident_id           varchar2(14) not null,
    name                  varchar2(10) not null,
    year                  int,
    address               varchar2(10),
    dept_id               varchar2(10),
    constraint pk_student primary key(stu_id)
)
```

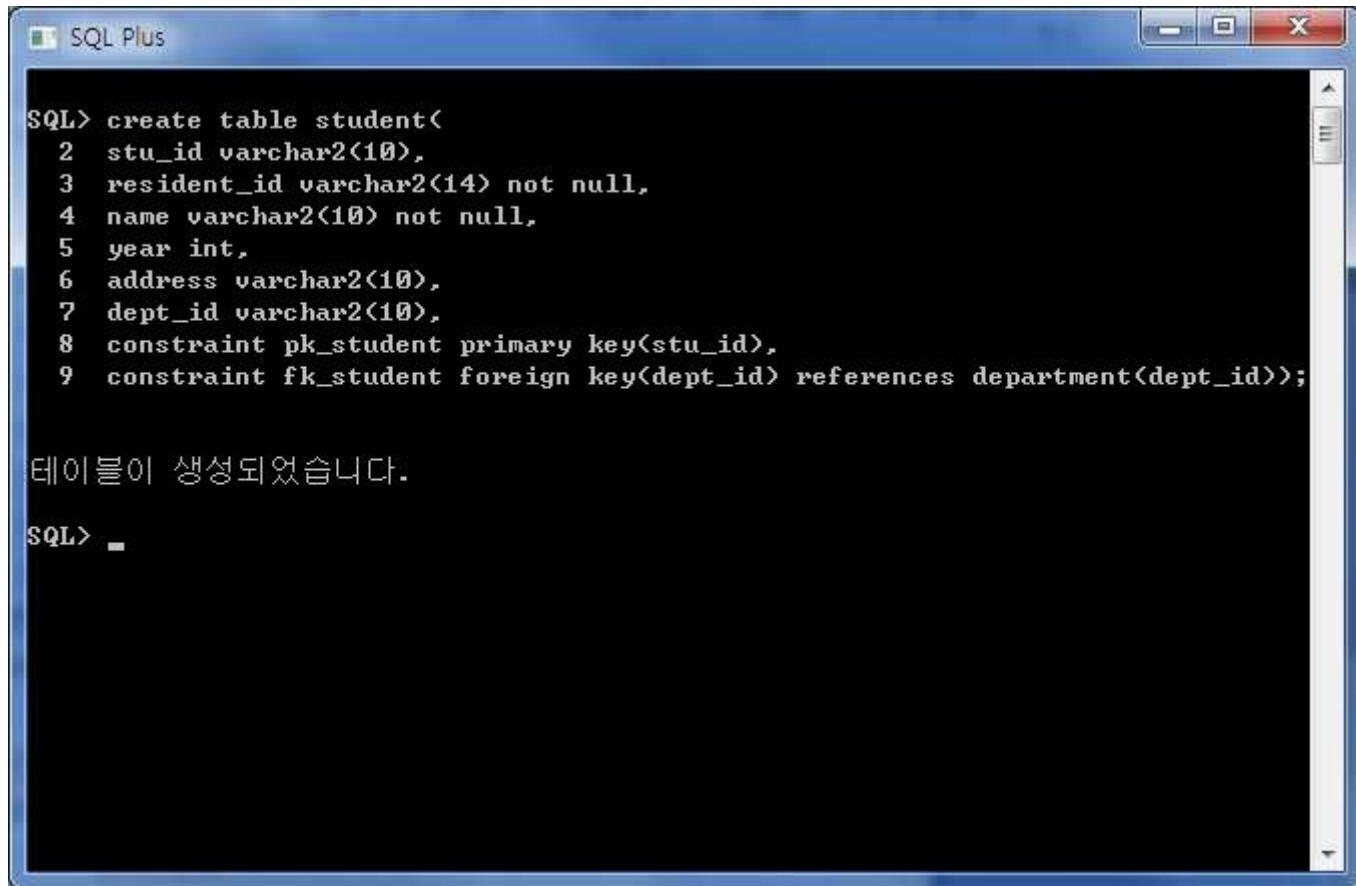
테이블 생성(student table)

- ▶ 외래키까지 포함된 student 테이블 생성 예

(질의 3)

```
create table student
(
    stu_id                varchar2(10),
    resident_id           varchar2(14) not null,
    name                  varchar2(10) not null,
    year                  int,
    address               varchar2(10),
    dept_id               varchar2(10),
    constraint pk_student primary key(stu_id),
    constraint fk_student foreign key(dept_id) references
                        department(dept_id)
)
```

테이블 생성(student table)

A screenshot of a SQL Plus window titled "SQL Plus". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main area is a black terminal with white text. It shows the SQL command to create a table named 'student' with columns: stu_id (varchar2(10)), resident_id (varchar2(14) not null), name (varchar2(10) not null), year (int), address (varchar2(10)), and dept_id (varchar2(10)). It also includes two constraints: a primary key constraint 'pk_student' on stu_id, and a foreign key constraint 'fk_student' on dept_id that references the department table's dept_id. Below the command, a message in Korean states "테이블이 생성되었습니다." (Table created successfully). The prompt "SQL> _" is visible at the bottom.

```
SQL> create table student(  
2  stu_id varchar2(10),  
3  resident_id varchar2(14) not null,  
4  name varchar2(10) not null,  
5  year int,  
6  address varchar2(10),  
7  dept_id varchar2(10),  
8  constraint pk_student primary key(stu_id),  
9  constraint fk_student foreign key(dept_id) references department(dept_id));  
  
테이블이 생성되었습니다.  
  
SQL> _
```

테이블 생성

▶ professor 테이블

(질의 5)

create table professor

(

prof_id

varchar2(10) ,

resident_id

varchar2(14)

not null,

name

varchar2(10)

not null,

dept_id

varchar2(10),

position

varchar2(10),

year_emp

int,

constraint

pk_professor

primary key(prof_id),

constraint

fk_professor

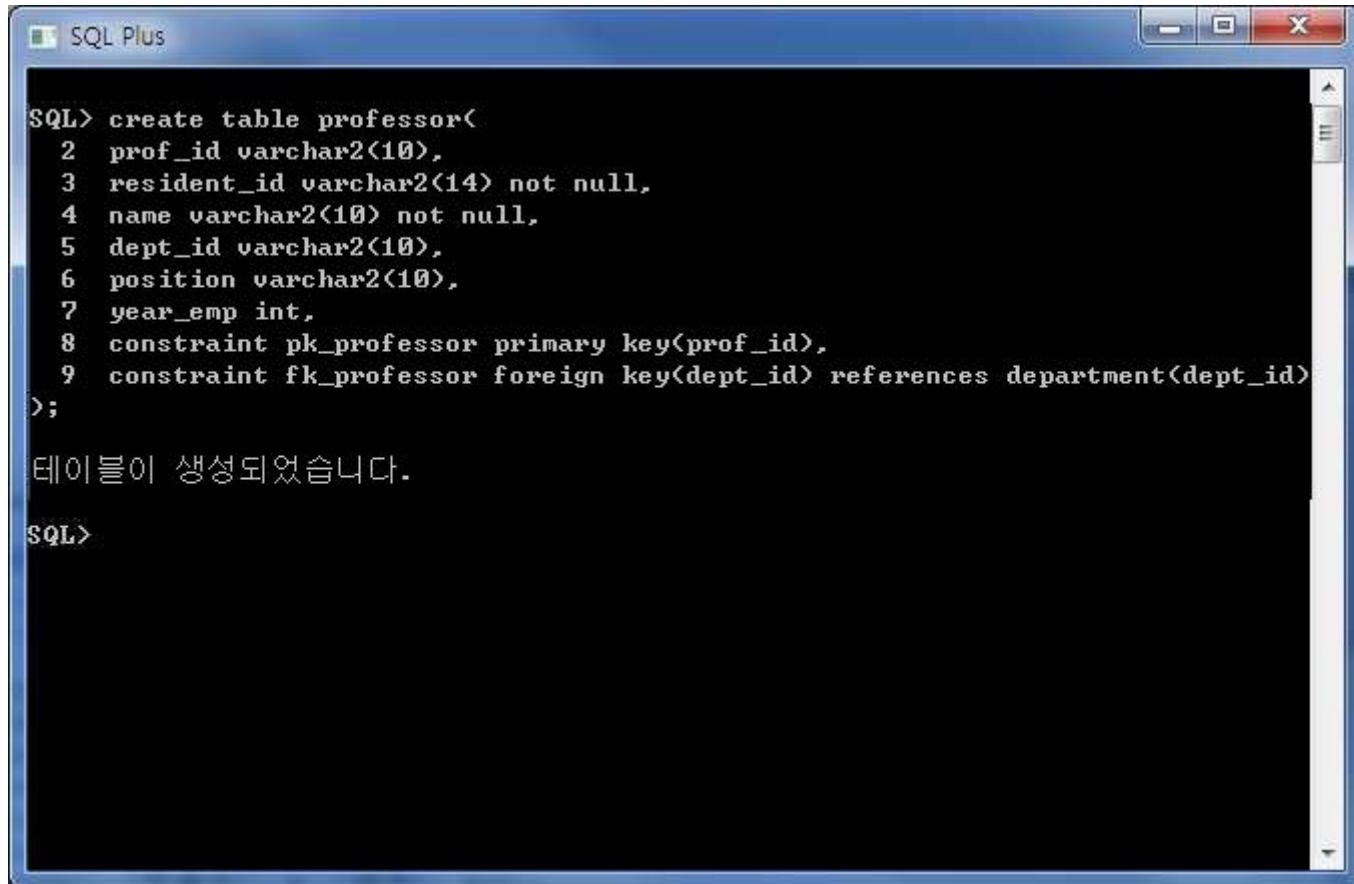
foreign key(dept_id)

references department(dept_id)

)

테이블 생성

▶ professor 테이블



```
SQL> create table professor(  
2  prof_id varchar2(10),  
3  resident_id varchar2(14) not null,  
4  name varchar2(10) not null,  
5  dept_id varchar2(10),  
6  position varchar2(10),  
7  year_emp int,  
8  constraint pk_professor primary key(prof_id),  
9  constraint fk_professor foreign key(dept_id) references department(dept_id)  
>);  
  
테이블이 생성되었습니다.  
  
SQL>
```

테이블 생성

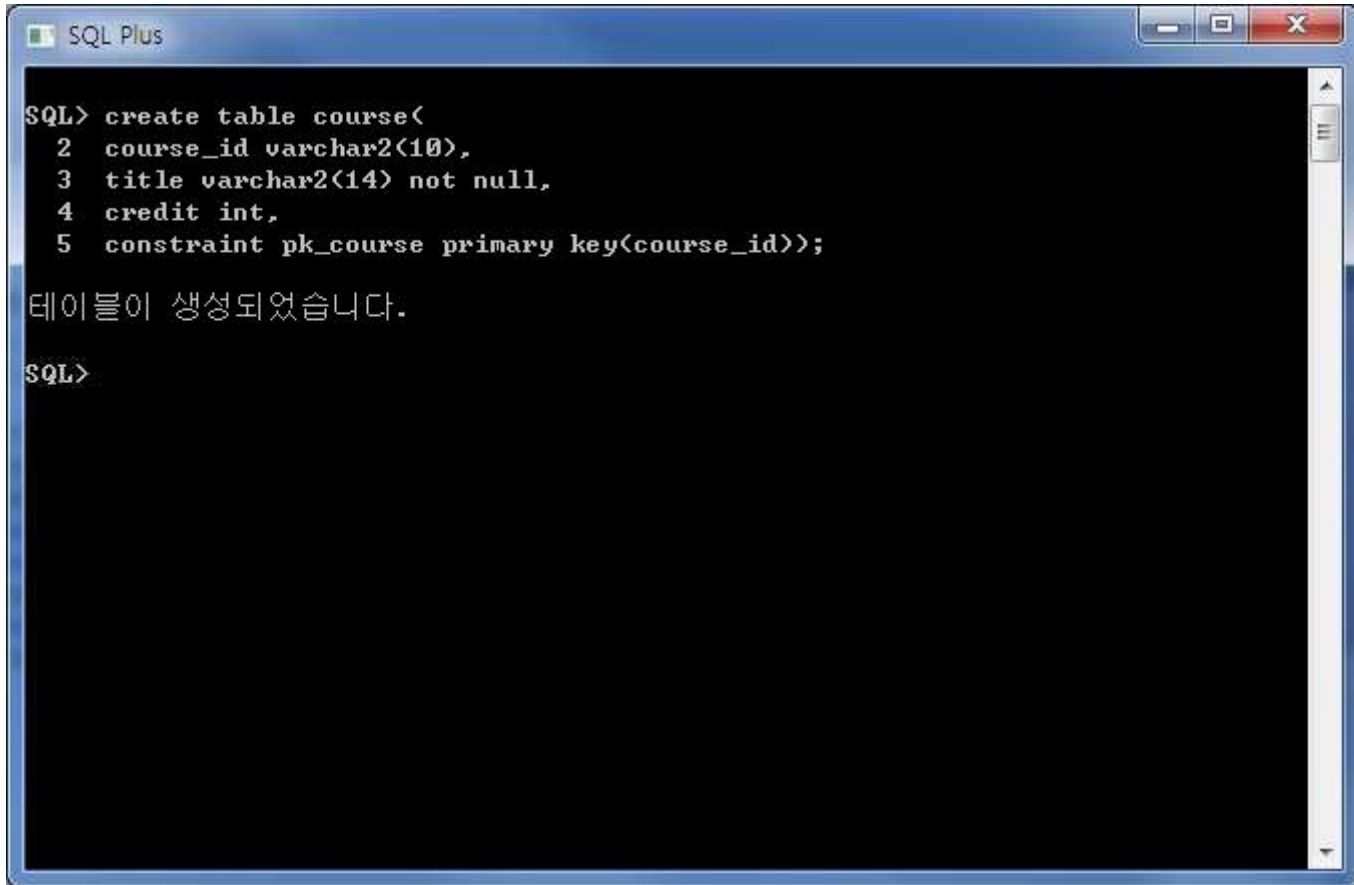
▶ course 테이블

(질의 6)

```
create table course
(
    course_id          varchar2(10) ,
    title              varchar2(14)  not null,
    credit              int,
    constraint pk_course primary key(course_id)
)
```

테이블 생성

▶ course 테이블

A screenshot of a SQL Plus window titled "SQL Plus". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main area is a black terminal with white text. The text shows the SQL command to create a table named 'course' with columns 'course_id', 'title', and 'credit', and a primary key constraint on 'course_id'. Below the command, a message in Korean states "테이블이 생성되었습니다." (Table created successfully). The prompt "SQL>" is visible at the bottom.

```
SQL> create table course(  
  2  course_id varchar2(10),  
  3  title varchar2(14) not null,  
  4  credit int,  
  5  constraint pk_course primary key(course_id));  
  
테이블이 생성되었습니다.  
  
SQL>
```

테이블 생성

▶ class 테이블

(질의 7)

create table class

(

class_id **varchar2(10) ,**

course_id **varchar2(10),**

year **int,**

semester **int,**

division **char(1),**

prof_id **varchar2(10),**

classroom **varchar2(9),**

enroll **int,**

constraint **pk_class** **primary key**(class_id),

constraint **fk_class1** **foreign key**(course_id)

constraint **references** **course**(course_id),

constraint **fk_class2** **foreign key**(prof_id)

constraint **references** **professor**(prof_id)

)

테이블 생성

▶ class 테이블



```
SQL> create table class(  
  2  class_id varchar2(10),  
  3  course_id varchar2(10),  
  4  year int,  
  5  semester int,  
  6  division char(1),  
  7  prof_id varchar2(10),  
  8  classroom varchar2(9),  
  9  enroll int,  
10  constraint pk_class primary key(class_id),  
11  constraint fk_class1 foreign key(course_id) references course(course_id),  
12  constraint fk_class2 foreign key(prof_id) references professor(prof_id));  
  
테이블이 생성되었습니다.  
  
SQL> _
```

테이블 생성

▶ takes 테이블

(질의 8)

```
create table takes
```

```
(
```

```
    stu_id
```

```
    class_id
```

```
    grade
```

```
    constraint
```

```
    constraint
```

```
    constraint
```

```
)
```

```
    varchar2(10) ,
```

```
    varchar2(10),
```

```
    varchar(5),
```

```
    pk_takes
```

```
    fk_takes1
```

```
    references
```

```
    fk_takes2
```

```
    references
```

```
    primary key(stu_id, class_id),
```

```
    foreign key(stu_id)
```

```
    student(stu_id),
```

```
    foreign key(class_id)
```

```
    class(class_id)
```

테이블 생성

▶ takes 테이블



```
SQL> create table takes(  
  2  stu_id varchar2(10),  
  3  class_id varchar2(10),  
  4  grade char(5),  
  5  constraint pk_takes primary key(stu_id, class_id),  
  6  constraint fk_takes1 foreign key(stu_id) references student(stu_id),  
  7  constraint fk_takes2 foreign key(class_id) references class(class_id));  
  
테이블이 생성되었습니다.  
  
SQL> _
```

테이블 삭제

▶ 형식

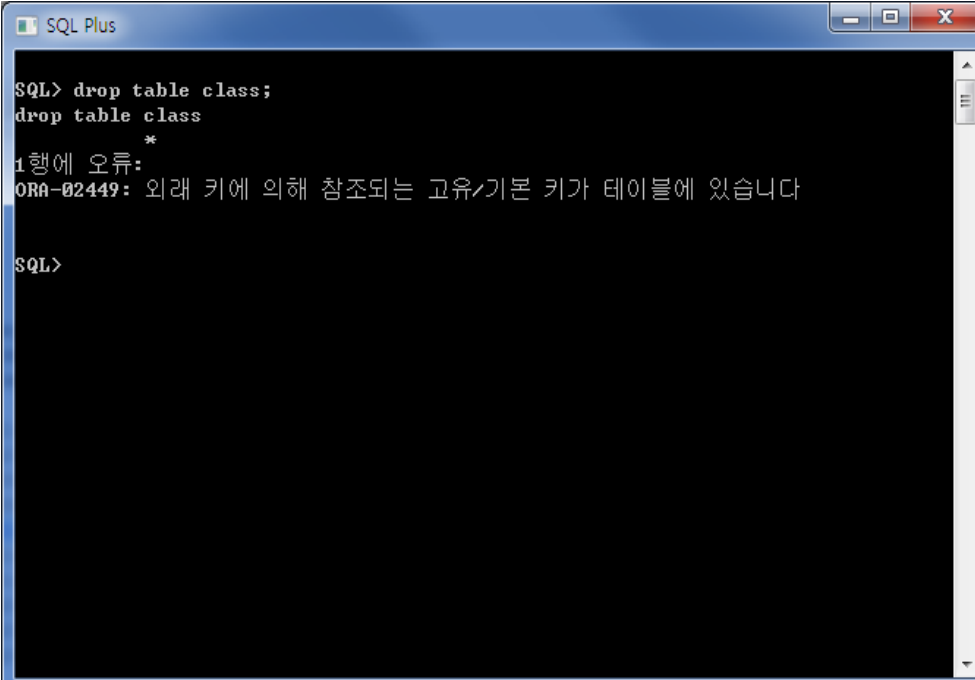
drop table <테이블이름>

▶ 주의

- ▶ 다른 테이블에서 외래키로 참조되는 경우에는 삭제할 수 없음

▶ 예

- ▶ class 테이블은 takes 테이블에서 외래키로 참조됨
- ▶ takes 테이블을 삭제하기 전에는 class 테이블을 삭제할 수 없음



```
SQL> drop table class;
drop table class
*
1행에 오류:
ORA-02449: 외래 키에 의해 참조되는 고유/기본 키가 테이블에 있습니다

SQL>
```

테이블 수정

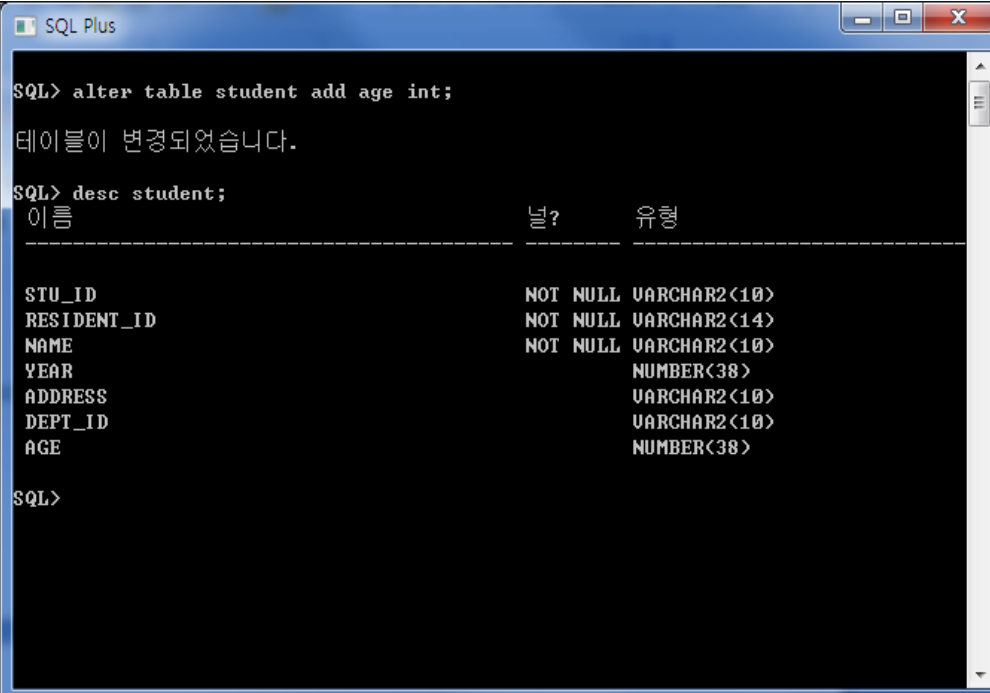
- ▶ 기존 테이블에 새로운 필드를 추가하거나 기존 필드를 삭제
- ▶ 필드 추가 형식

alter table <테이블이름> **add** <추가할필드>

- ▶ 예) student 테이블에 age 필드를 추가

(질의 9)

alter table student
add age int



The screenshot shows a SQL Plus window with the following content:

```
SQL> alter table student add age int;
테이블이 변경되었습니다.

SQL> desc student;
  이름                널?       유형
-----
STU_ID                NOT NULL  VARCHAR2(10)
RESIDENT_ID           NOT NULL  VARCHAR2(14)
NAME                  NOT NULL  VARCHAR2(10)
YEAR                  NUMBER(38)
ADDRESS                VARCHAR2(10)
DEPT_ID                VARCHAR2(10)
AGE                    NUMBER(38)
```

The window title is "SQL Plus". The output of the first command is "테이블이 변경되었습니다." (Table modified). The output of the second command is a table description showing the columns and their data types.

테이블 수정

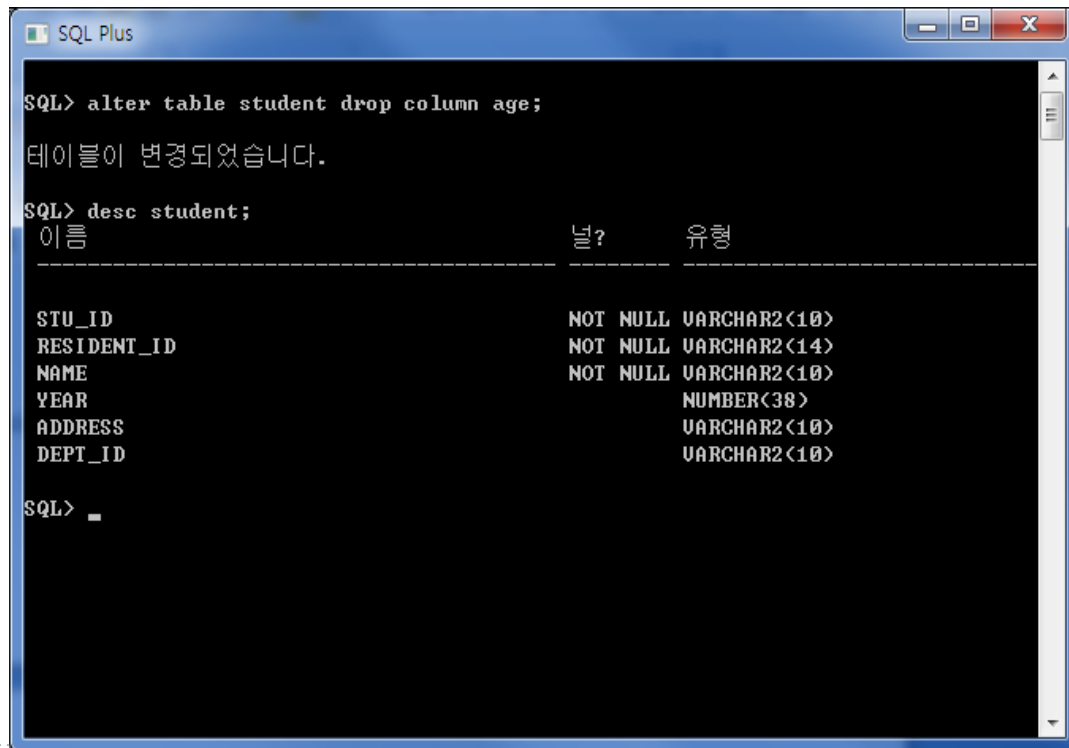
▶ 필드 삭제 형식

```
alter table <테이블이름> drop column <삭제할필드>
```

▶ 예)

(질의 10)

```
alter table student  
drop column age
```



```
SQL Plus

SQL> alter table student drop column age;

테이블이 변경되었습니다.

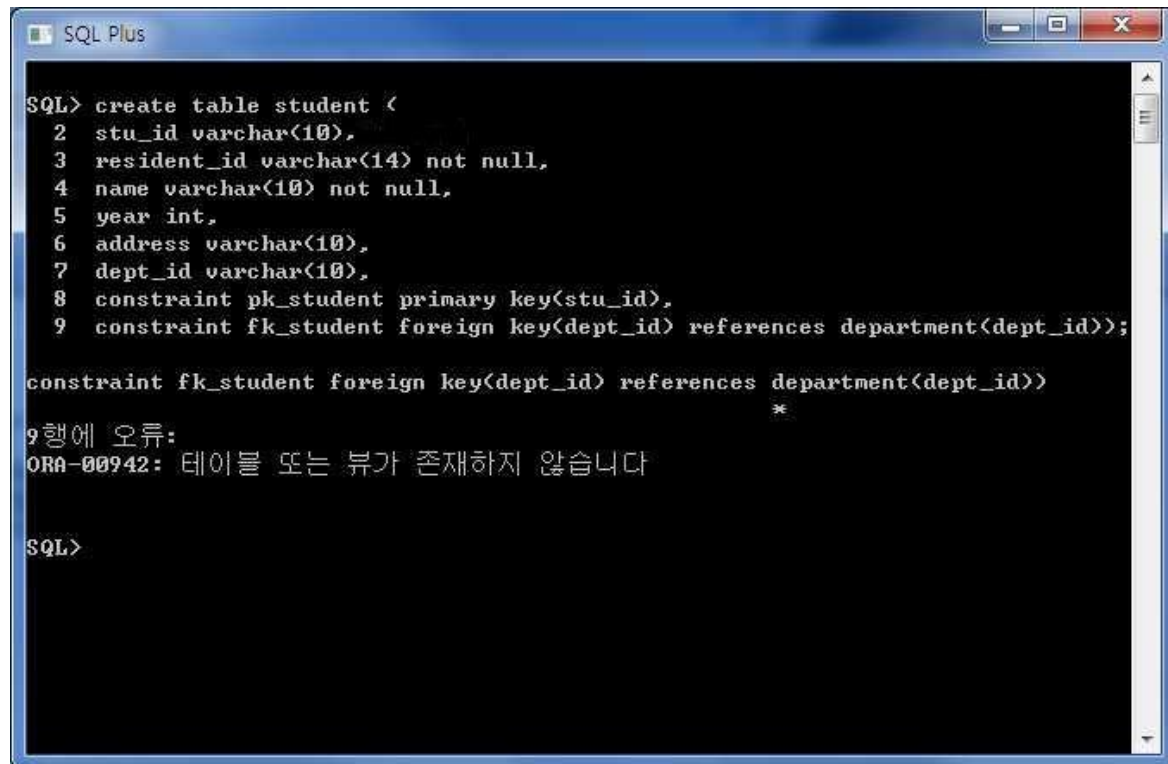
SQL> desc student;

이름?          유형
-----
STU_ID          NOT NULL VARCHAR2(10)
RESIDENT_ID     NOT NULL VARCHAR2(14)
NAME            NOT NULL VARCHAR2(10)
YEAR            NUMBER(38)
ADDRESS          VARCHAR2(10)
DEPT_ID          VARCHAR2(10)

SQL> _
```

기본키, 외래키 관련 주의사항

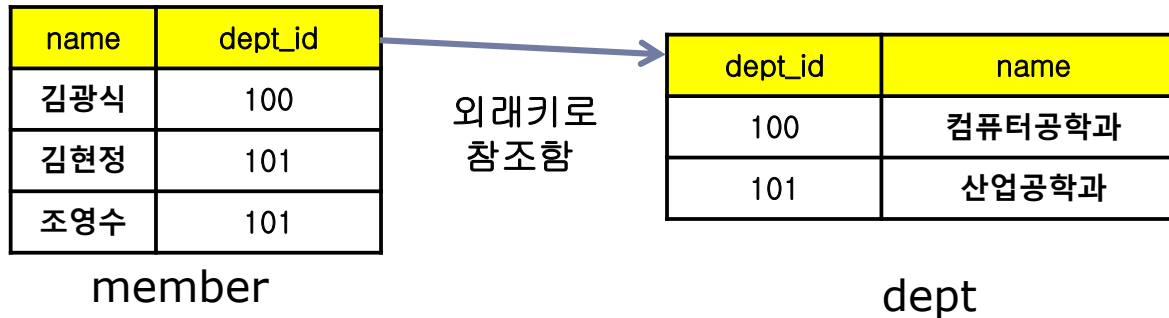
- ▶ 외래키를 필드로 갖는 테이블을 생성 할 때
 - ▶ 외래키가 참조하는 테이블을 먼저 생성
- ▶ 예) 만일 department 테이블이 존재하지 않는 상태에서 student 테이블을 먼저 생성하면 다음과 같은 오류발생

A screenshot of a SQL Plus window titled "SQL Plus". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main area is a black terminal with white text. The text shows an SQL command to create a table named 'student' with several columns and constraints. The command is: SQL> create table student (
2 stu_id varchar(10),
3 resident_id varchar(14) not null,
4 name varchar(10) not null,
5 year int,
6 address varchar(10),
7 dept_id varchar(10),
8 constraint pk_student primary key(stu_id),
9 constraint fk_student foreign key(dept_id) references department(dept_id));
Below the command, there is an error message: 9행에 오류:
ORA-00942: 테이블 또는 뷰가 존재하지 않습니다
The prompt SQL> is visible at the bottom of the terminal area.

```
SQL> create table student (  
2 stu_id varchar(10),  
3 resident_id varchar(14) not null,  
4 name varchar(10) not null,  
5 year int,  
6 address varchar(10),  
7 dept_id varchar(10),  
8 constraint pk_student primary key(stu_id),  
9 constraint fk_student foreign key(dept_id) references department(dept_id));  
  
constraint fk_student foreign key(dept_id) references department(dept_id)  
*  
9행에 오류:  
ORA-00942: 테이블 또는 뷰가 존재하지 않습니다  
  
SQL>
```

기본키, 외래키 관련 주의사항

- ▶ 테이블을 삭제할 때도 같은 문제가 발생함
- ▶ 예)



- ▶ member 테이블이 있는 한 dept 테이블을 삭제할 수 없음
- ▶ dept 테이블을 삭제하려면 member 테이블을 먼저 삭제하던지, 외래키를 해제해야 함
 - ▶ 외래키 해제는 5장에서 다룸

데이터 조작 언어

- ▶ 레코드 삽입
- ▶ 레코드 수정
- ▶ 레코드 삭제
- ▶ 레코드 검색

레코드 삽입

▶ 형식

insert into <테이블이름> (<필드리스트>) **values** (<값리스트>)

- ▶ <필드리스트>
 - ▶ 삽입에 사용될 테이블의 필드들
- ▶ <값리스트>
 - ▶ <필드리스트>의 순서에 맞춰 삽입될 값
- ▶ <필드리스트>에 나열되지 않은 필드에 대해서는 널값이 입력됨
- ▶ <필드리스트>를 생략할 경우 <값리스트>에는 테이블을 생성할 때 나열한 필드의 순서에 맞춰서 값을 나열

레코드 삽입

▶ 예

(질의 11)

```
insert into department (dept_id, dept_name, office)
values ('920', '컴퓨터공학과', '201호')
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> insert into department (dept_id, dept_name, office)
2 values('920', '컴퓨터공학과', '201호');

1 개의 행이 만들어졌습니다.

SQL> select * from department;
```

DEPT_ID	DEPT_NAME	OFFICE
920	컴퓨터공학과	201호

SQL>

입력확인을 위한
SQL 명령

레코드 삽입

- ▶ 삽입 명령문에 필드 이름을 나열할 경우 그 순서는 테이블을 생성할 때 지정한 순서와 반드시 일치할 필요는 없음
- ▶ 예) (질의 11)과 동일한 SQL

(질의 12)

```
insert into department (office, dept_id, dept_name)
values ('201호', '920', '컴퓨터공학과')
```

- ▶ department 테이블의 필드들 중에서 office 필드를 생략하는 경우
 - ▶ 생략된 필드에는 널이 입력

(질의 13)

```
insert into department (dept_id, dept_name)
values ('920', '컴퓨터공학과')
```

- ▶ 단, **not null**로 설정된 필드는 널 값이 들어갈 수 없는 필드이기 때문에 **insert**문의 <필드리스트>에서 생략할 수 없음

레코드 삽입

- ▶ <필드리스트>를 사용하지 않고 데이터를 삽입하는 예

(질의 14)

**insert into
values**

department
('923', '산업공학과', '207호')

학사 데이터베이스의 데이터 삽입 예

```
insert into department values('920', '컴퓨터공학과', '201호')
```

```
insert into department values('923', '산업공학과', '207호')
```

```
insert into department values('925', '전자공학과', '308호')
```

```
insert into student
```

```
    values('1292001', '900424-1825409', '김광식', 3, '서울', '920')
```

```
insert into student
```

```
    values('1292002', '900305-1730021', '김정현', 3, '서울', '920')
```

```
insert into student
```

```
    values('1292003', '891021-2308302', '김현정', 4, '대전', '920')
```

```
insert into student
```

```
    values('1292301', '890902-2704012', '김현정', 2, '대구', '923')
```

```
insert into student
```

```
    values('1292303', '910715-1524390', '박광수', 3, '광주', '923')
```

```
insert into student
```

```
    values('1292305', '921011-1809003', '김우주', 4, '부산', '923')
```

```
insert into student
```

```
    values('1292501', '900825-1506390', '박철수', 3, '대전', '925')
```

```
insert into student
```

```
    values('1292502', '911011-1809003', '백태성', 3, '서울', '925')
```

학사 데이터베이스의 데이터 삽입 예

```
insert into professor
  values('92001', '590327-1839240', '이태규', '920', '교수', 1997)
insert into professor
  values('92002', '690702-1350026', '고희석', '920', '부교수', 2003)
insert into professor
  values('92301', '741011-2765501', '최성희', '923', '부교수', 2005)
insert into professor
  values('92302', '750728-1102458', '김태석', '923', '교수', 1999)
insert into professor
  values('92501', '620505-1200546', '박철재', '925', '조교수', 2007)
insert into professor
  values('92502', '740101-1830264', '장민석', '925', '부교수', 2005)
insert into course values('C101', '전산개론', 3)
insert into course values('C102', '자료구조', 3)
insert into course values('C103', '데이터베이스', 4)
insert into course values('C301', '운영체제', 3)
insert into course values('C302', '컴퓨터구조', 3)
insert into course values('C303', '이산수학', 4)
insert into course values('C304', '객체지향언어', 4)
insert into course values('C501', '인공지능', 3)
insert into course values('C502', '알고리즘', 2)
```

학사 데이터베이스의 데이터 삽입 예

```
insert into class values('C101-01', 'C101', 2012, 1, 'A', '92301', '301호', 40)
insert into class values('C102-01', 'C102', 2012, 1, 'A', '92001', '209호', 30)
insert into class values('C103-01', 'C103', 2012, 1, 'A', '92501', '208호', 30)
insert into class values('C103-02', 'C103', 2012, 1, 'B', '92301', '301호', 30)
insert into class values('C501-01', 'C501', 2012, 1, 'A', '92501', '103호', 45)
insert into class values('C501-02', 'C501', 2012, 1, 'B', '92502', '204호', 25)
insert into class values('C301-01', 'C301', 2012, 2, 'A', '92502', '301호', 30)
insert into class values('C302-01', 'C302', 2012, 2, 'A', '92501', '209호', 45)
insert into class values('C502-01', 'C502', 2012, 2, 'A', '92001', '209호', 30)
insert into class values('C502-02', 'C502', 2012, 2, 'B', '92301', '103호', 26)
```


학사 데이터베이스의 데이터 삽입 예

```
insert into takes values('1292001', 'C101-01', 'B+')
insert into takes values('1292001', 'C103-01', 'A+')
insert into takes values('1292001', 'C301-01', 'A')
insert into takes values('1292002', 'C102-01', 'A')
insert into takes values('1292002', 'C103-01', 'B+')
insert into takes values('1292002', 'C502-01', 'C+')
insert into takes values('1292003', 'C103-02', 'B')
insert into takes values('1292003', 'C501-02', 'A+')
insert into takes values('1292301', 'C102-01', 'C+')
insert into takes values('1292303', 'C102-01', 'C')
insert into takes values('1292303', 'C103-02', 'B+')
insert into takes values('1292303', 'C501-01', 'A+')
```

레코드 수정

▶ 형식

```
update <테이블이름>  
set <수정내역>  
where <조건>
```

▶ <수정내역>

- ▶ 대상 테이블의 필드에 들어가는 값을 수정하기 위한 산술식
- ▶ ‘;’를 이용해서 여러 필드에 대한 수정 내역을 지정

▶ <조건>

- ▶ 대상이 되는 레코드에 대한 조건을 기술
- ▶ 관계대수에서 선택 연산의 조건식과 같은 의미
- ▶ 테이블의 모든 레코드에 대해 수정을 적용하려면 **where** 절을 생략

레코드 수정

- ▶ 예) student 테이블에서 모든 학생들의 학년을 하나씩 증가

(질의 16)

```
update student
set year = year + 1
```

- ▶ 예) professor 테이블에서 '고희석' 교수의 직위를 '교수'로 수정하고 학과번호를 '923'으로 수정

(질의 17)

```
update professor
set position='교수', dept_id='923'
where name='고희석'
```

레코드 삭제

▶ 형식

delete from	<테이블이름>
where	<조건>

- ▶ **where**절에 지정된 조건을 만족하는 레코드를 삭제
- ▶ **where**절이 생략되면 테이블에서 모든 레코드를 삭제
- ▶ 예) professor 테이블에서 이름이 '김태석'인 교수를 삭제

(질의 18)

delete from	professor
where	name='김태석'

- ▶ **delete**문을 이용하여 테이블의 모든 레코드를 삭제하더라도 테이블은 삭제되지 않음

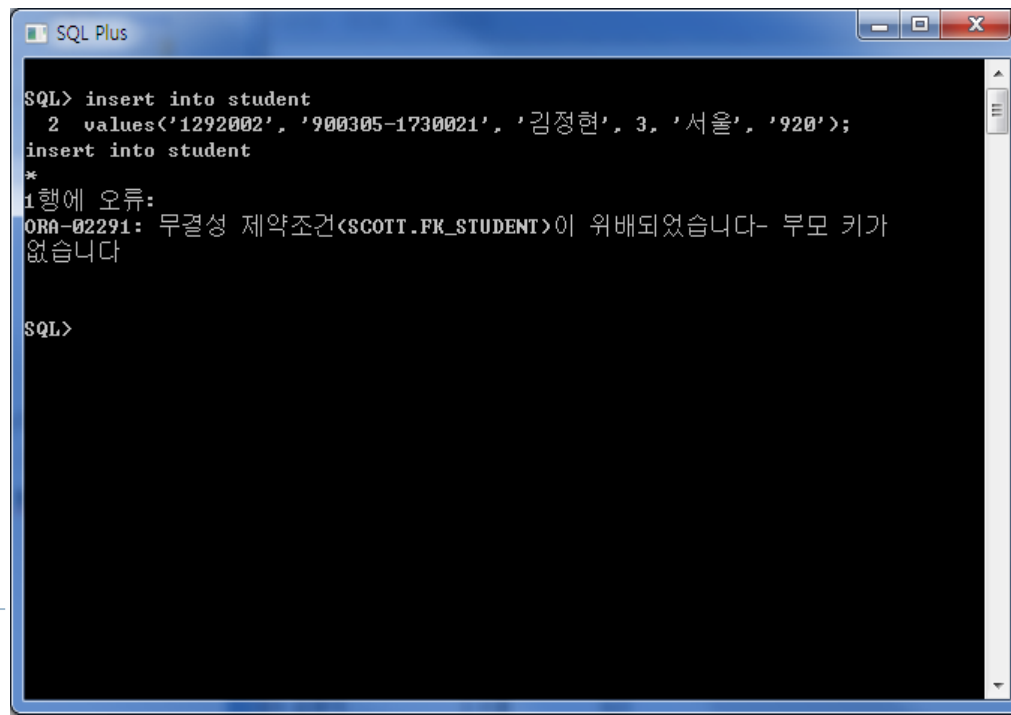
레코드 삽입 시 주의사항

- ▶ 외래키로 사용되는 필드에 대해 데이터를 삽입할 때
 - ▶ 참조하는 테이블의 해당 필드에 그 값을 먼저 삽입해야 함
 - ▶ 예) department 테이블이 생성되긴 했지만 아직 레코드가 삽입되지 않은 상태에서 다음질의의 실행 결과

(질의 19)

**insert into
values**

student
('1292002', '900305-1730021', '김정현', 3, '서울', '920')

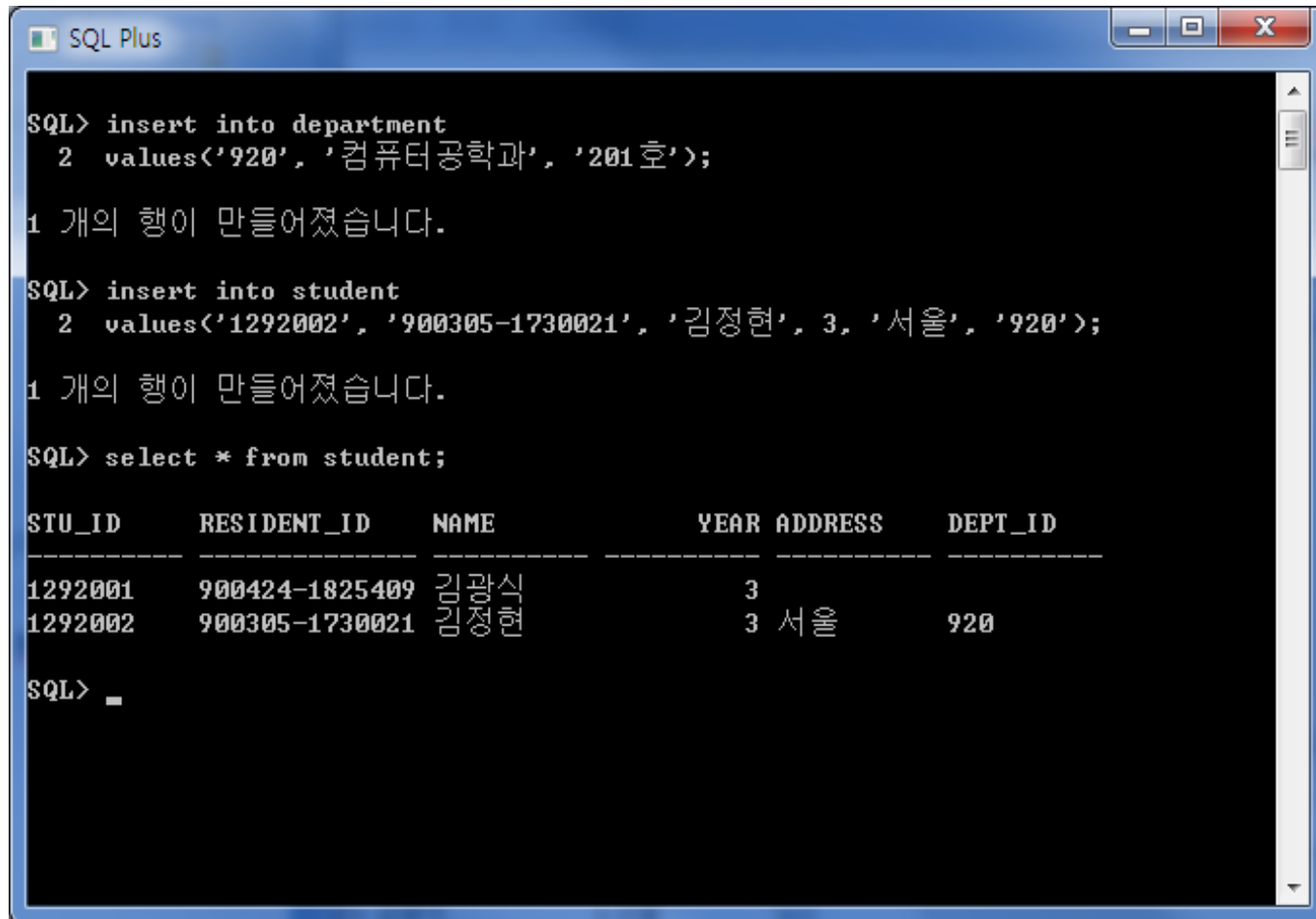


```
SQL> insert into student
2 values('1292002', '900305-1730021', '김정현', 3, '서울', '920');
insert into student
*
1행에 오류:
ORA-02291: 무결성 제약조건(SCOTT.FK_STUDENT)이 위배되었습니다- 부모 키가
없습니다

SQL>
```

레코드 삽입 시 주의사항

- ▶ department 테이블에 dept_id 필드의 값이 '920'인 레코드가 먼저 삽입



```
SQL> insert into department
  2  values('920', '컴퓨터공학과', '201호');

1 개의 행이 만들어졌습니다.

SQL> insert into student
  2  values('1292002', '900305-1730021', '김정현', 3, '서울', '920');

1 개의 행이 만들어졌습니다.

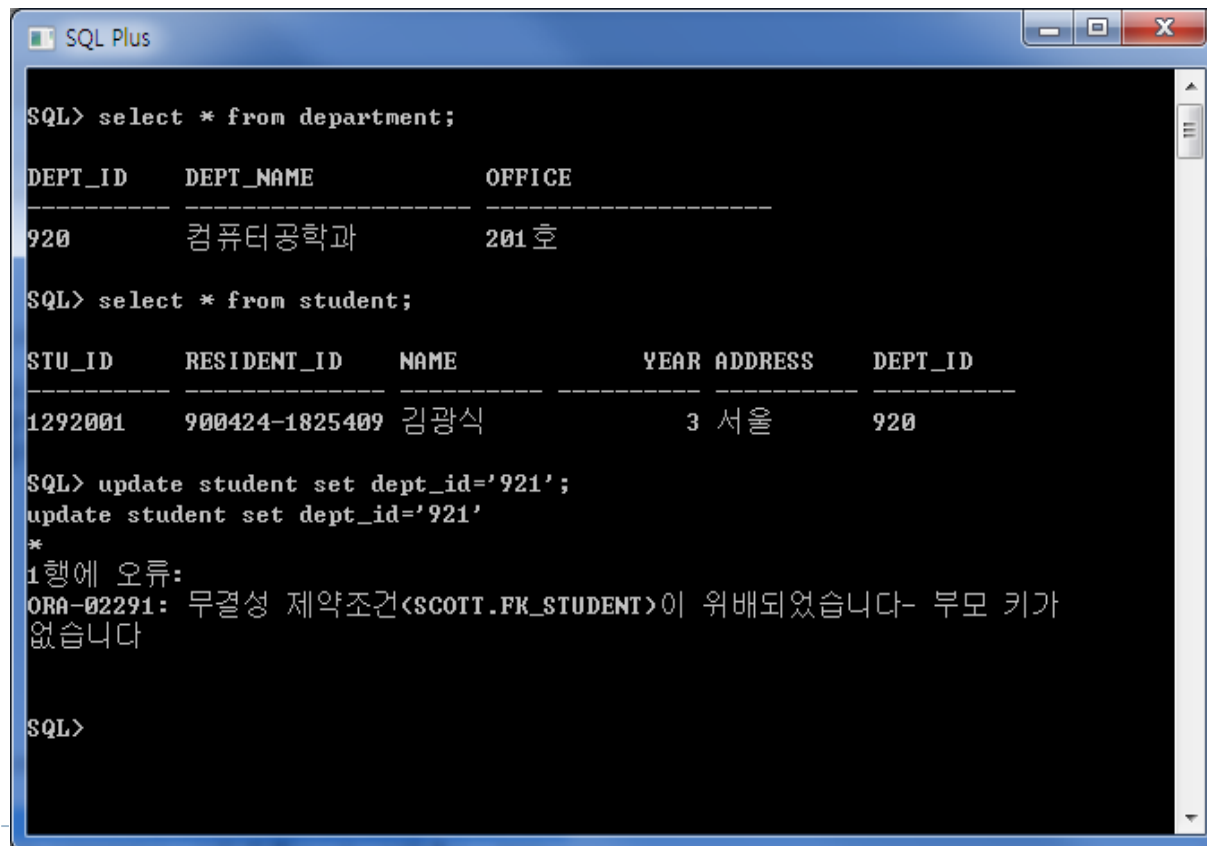
SQL> select * from student;
```

STU_ID	RESIDENT_ID	NAME	YEAR	ADDRESS	DEPT_ID
1292001	900424-1825409	김광식	3		
1292002	900305-1730021	김정현	3	서울	920

```
SQL> _
```

레코드 수정 시 주의사항

- ▶ 외래키로 사용되는 필드의 값을 수정할 때
 - ▶ 외래키가 참조하는 테이블에 삽입되어 있는 값으로만 수정이 가능
 - ▶ 예) department 테이블에 dept_id 필드의 값이 '920'인 레코드만 삽입되어 있는 상황



```
SQL> select * from department;

DEPT_ID   DEPT_NAME      OFFICE
-----
920       컴퓨터공학과   201호

SQL> select * from student;

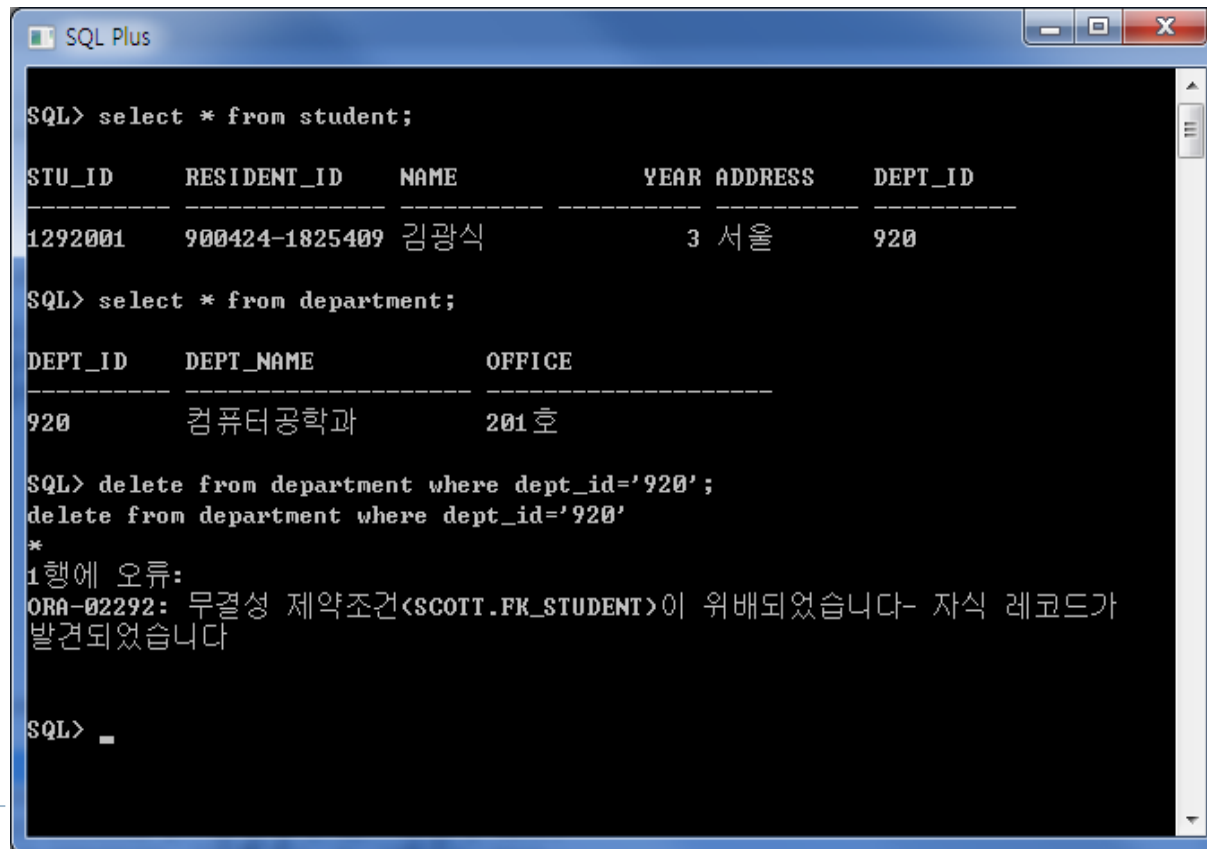
STU_ID    RESIDENT_ID    NAME                YEAR ADDRESS    DEPT_ID
-----
1292001   900424-1825409 김광식              3 서울      920

SQL> update student set dept_id='921';
update student set dept_id='921'
*
1행에 오류:
ORA-02291: 무결성 제약조건(SCOTT.FK_STUDENT)이 위반되었습니다- 부모 키가
없습니다

SQL>
```

레코드 삭제 시 주의사항

- ▶ 외래키로 참조되는 필드를 가지고 있는 테이블에서 레코드를 삭제할 경우에도 오류가 발생할 수 있음
 - ▶ student 테이블에서 외래키로 참조하는 department 테이블의 레코드에 대한 삭제 시도



```
SQL> select * from student;

STU_ID    RESIDENT_ID    NAME                YEAR ADDRESS    DEPT_ID
-----
1292001    900424-1825409 김광식                3 서울        920

SQL> select * from department;

DEPT_ID    DEPT_NAME        OFFICE
-----
920        컴퓨터공학과        201호

SQL> delete from department where dept_id='920';
delete from department where dept_id='920'
*
1행에 오류:
ORA-02292: 무결성 제약조건(SCOTT.FK_STUDENT)이 위배되었습니다- 자식 레코드가 발견되었습니다

SQL> _
```


레코드 검색

▶ SQL에서 **가장 많이 사용하고, 중요하며, 복잡함**

▶ 종류

- ▶ 기본 구조
- ▶ 재명명 연산
- ▶ LIKE 연산자
- ▶ 집합 연산
- ▶ 외부조인
- ▶ 집계 함수
- ▶ 널의 처리
- ▶ 중첩 질의

기본 구조

▶ 형식

select	<필드리스트>
from	<테이블리스트>
where	<조건>

▶ select

- ▶ 질의 결과로 출력할 필드들의 리스트, 관계대수의 추출연산에

▶ from

- ▶ 질의 실행과정에 필요한 테이블들의 리스트를, 관계대수의 카티션 프로덕트

▶ where

- ▶ 검색되어야 하는 레코드에 대한 조건, 관계대수의 선택연산에서
- ▶ 생략 가능

기본 구조

▶ 예)

(질의 20)

```
select  name, dept_name
from    department, student
where   department.dept_id = student.dept_id
```

▶ 의미

- ▶ from절에 나열된 department 테이블과 student 테이블을 카티션 프로덕트
- ▶ where절에 지정된 조건식을 만족하는 레코드만 선택
 - ▶ 같은 이름의 필드가 두 개 이상의 테이블에 나타날 때 혼동을 피하기 위해 '테이블이름.필드이름'으로 표현
- ▶ 최종적으로 name 필드와 dept_name 필드의 값만을 추출하라

▶ 다음과 같은 의미

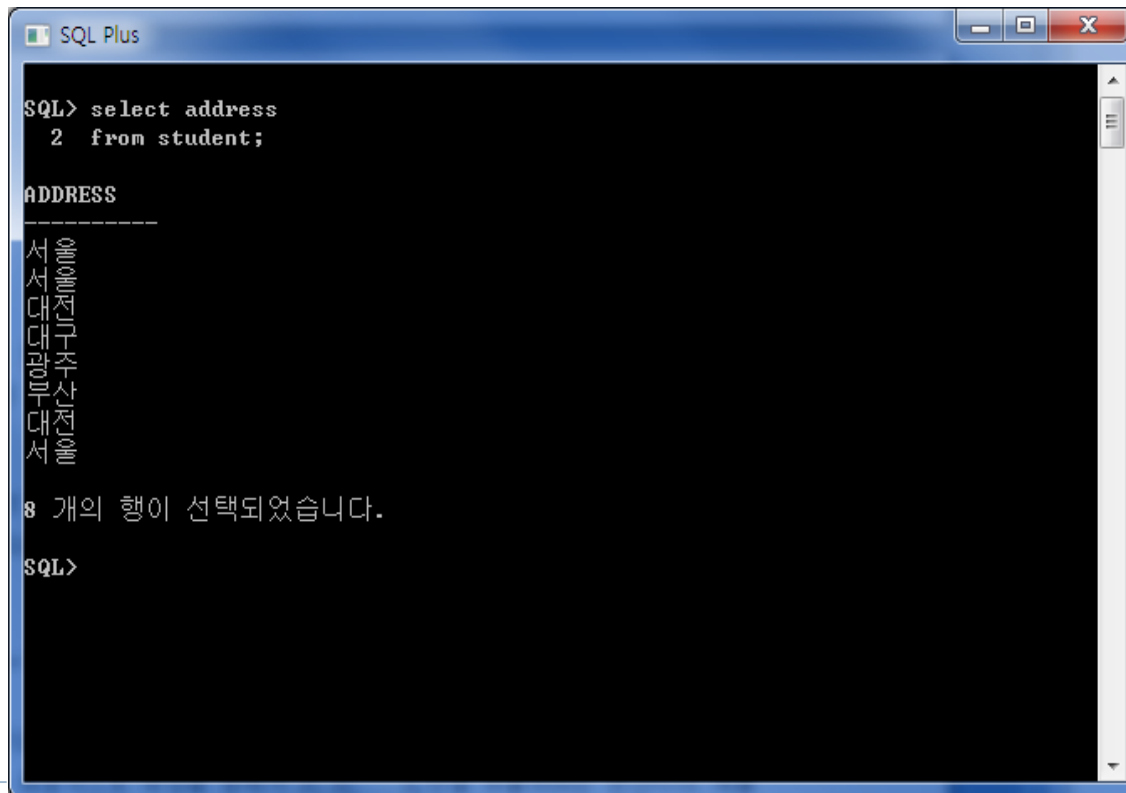
$$\pi_{name, dept_name} (\sigma_{department.dept_id = student.dept_id} (student \times department))$$

기본 구조

- ▶ 예) student 테이블에서 모든 학생들의 주소를 추출

(질의 21)

```
select address
from student
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select address
2  from student;
```

ADDRESS

서울
서울
서울
대전
대구
광주
부산
전주
서울

8 개의 행이 선택되었습니다.

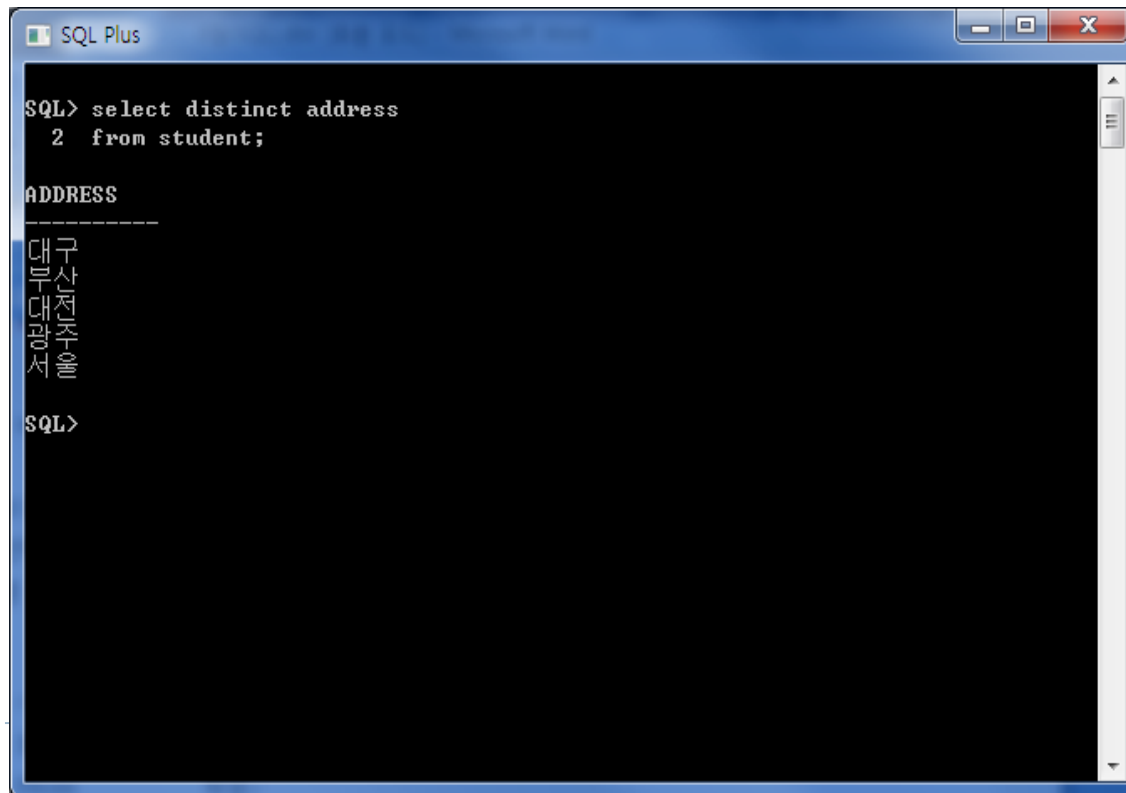
SQL>

기본 구조

- ▶ 중복된 레코드를 제거하고 검색하려면 distinct를 사용
- ▶ 예) student 테이블에서 모든 학생들의 주소를 추출

(질의 22)

```
select distinct address
from student
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select distinct address
2  from student;
```

The output is displayed as follows:

```
ADDRESS
-----
대구
부산
대전
광주
서울
```

The SQL prompt is visible at the bottom of the window.

기본 구조

- ▶ from 절에 나타난 테이블에서 모든 필드의 값을 추출할 경우에는 select 절에 모든 필드를 명시할 필요 없이 '*'를 사용
- ▶ 예) student 테이블에서 모든 레코드의 모든 필드 값을 추출

(질의 23)

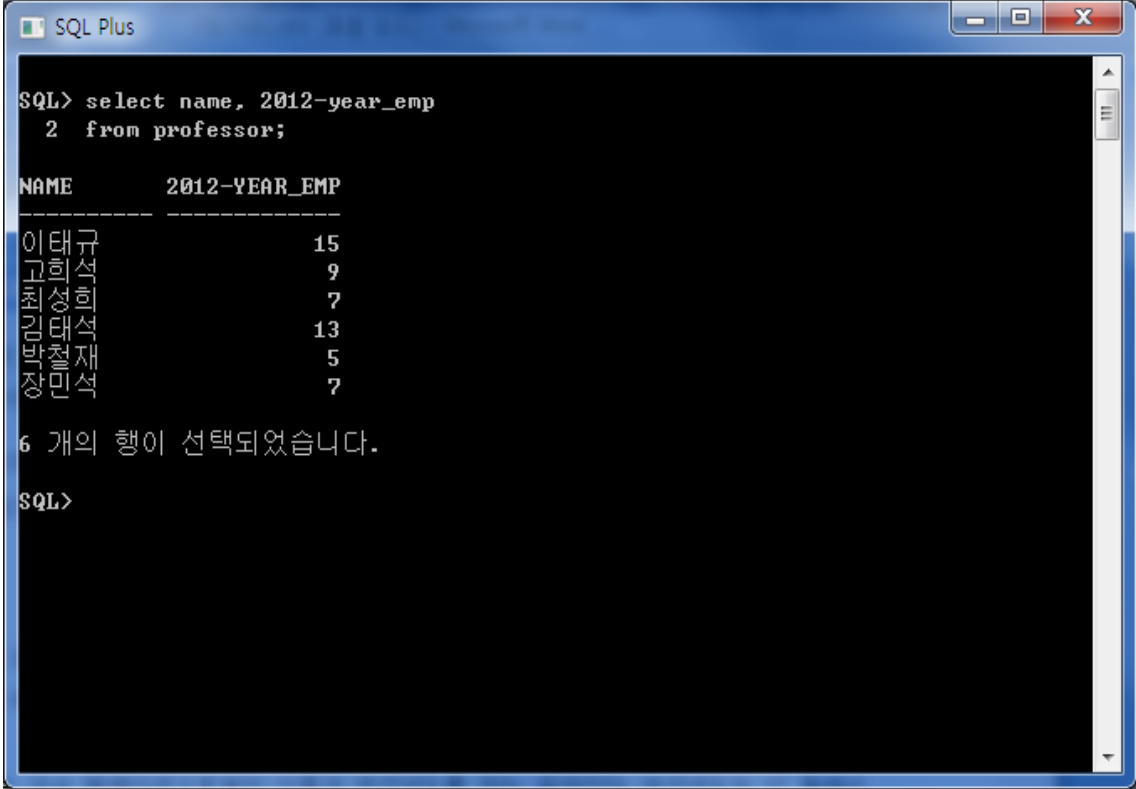
```
select *  
from student
```

기본 구조

- ▶ select절에 필드이름 외에 산술식이나 상수의 사용이 가능
- ▶ 예) professor 테이블에서 교수의 이름과 현재까지의 재직연수를 검색

(질의 24)

```
select name, 2012-year_emp  
from professor
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select name, 2012-year_emp  
2 from professor;
```

NAME	2012-YEAR_EMP
이태규	15
고희석	9
최성희	7
김태석	13
박철재	5
장민석	7

6 개의 행이 선택되었습니다.

```
SQL>
```

기본 구조

- ▶ from 절에 두 개 이상의 테이블이 포함된 질의
- ▶ 예) select문은 학생들의 이름, 학번, 그리고 소속 학과의 이름을 검색

(질의 25)

```
select    student.name, student.stu_id, department.dept_name
from      student, department
where     student.dept_id = department.dept_id
```


기본 구조

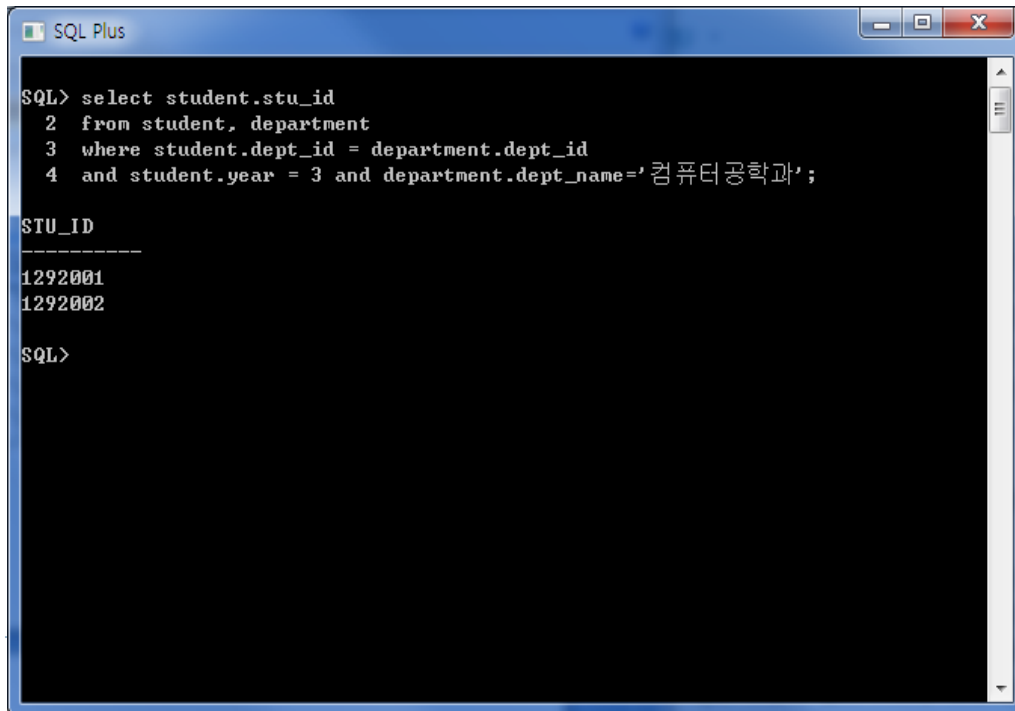
- ▶ from 절에 두 개 이상의 테이블을 포함하는 질의
 - ▶ 특정 조건이 없는 순수한 카티션 프러덕트보다는 테이블 간의 레코드에 대한 관계가 명시된 조인이나 자연조인이 대부분
 - ▶ 조인 질의
 - ▶ from 절에 두 개 이상의 테이블을 포함하는 질의

기본 구조

- ▶ 예) 컴퓨터공학과 3학년 학생들의 학번을 검색

(질의 26)

```
select    student.stu_id
from      student, department
where     student.dept_id = department.dept_id and
           student.year = 3 and
           department.dept_name='컴퓨터공학과'
```

A screenshot of a SQL Plus window titled "SQL Plus". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main area is black with white text. The text shows a SQL query being entered and executed. The query is:
SQL> select student.stu_id
2 from student, department
3 where student.dept_id = department.dept_id
4 and student.year = 3 and department.dept_name='컴퓨터공학과';
The results are displayed below the query:
STU_ID

1292001
1292002
The prompt "SQL>" is visible at the bottom of the window.

```
SQL Plus
SQL> select student.stu_id
2  from student, department
3  where student.dept_id = department.dept_id
4  and student.year = 3 and department.dept_name='컴퓨터공학과';

STU_ID
-----
1292001
1292002
SQL>
```

레코드의 순서 지정(order by)

- ▶ 검색 결과를 정렬하여 출력하는 기능
- ▶ select문 맨 마지막에 다음과 같은 order by절을 추가
- ▶ 형식

order by <필드리스트>

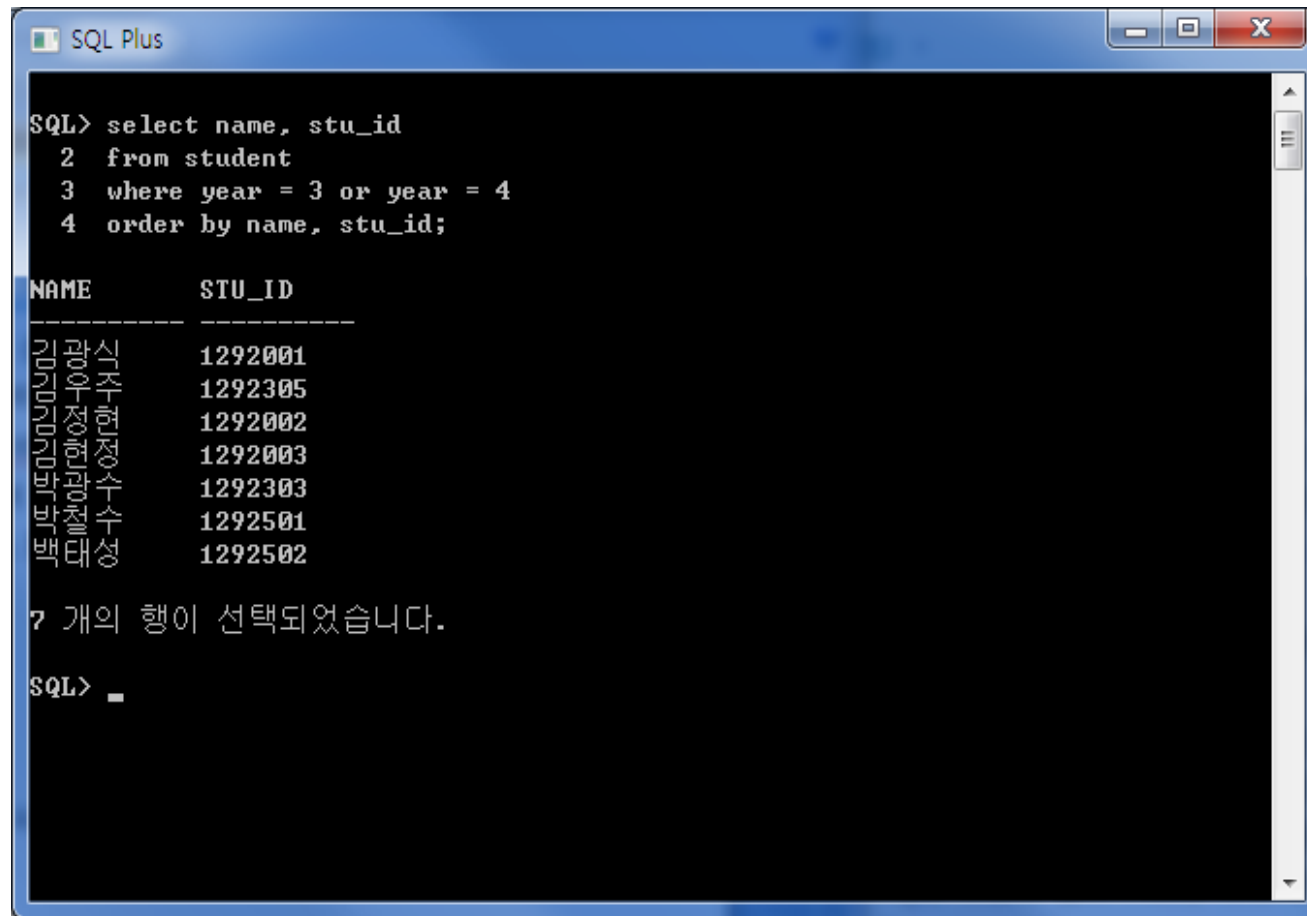
- ▶ 오름차순을 기본으로 하며 <필드리스트>에 여러 개의 필드를 나열할 경우 나열된 순서대로 정렬
- ▶ 예) student 테이블에서 3,4학년 학생들의 이름과 학번을 검색

(질의 27)

```
select    name, stu_id
from      student
where     year = 3 or year = 4
order by  name, stu_id
```

- ▶ 학생 이름(name 필드)으로 오름차순으로 정렬하고 같은 이름에 대해서는 학번의 오름차순으로 정렬

레코드의 순서 지정



The screenshot shows a SQL Plus window with a blue title bar. The command prompt shows a query to select names and student IDs from the 'student' table, filtered by year 3 or 4, and ordered by name and student ID. The results are displayed in a table with two columns: NAME and STU_ID. The names are listed vertically on the left, and the corresponding student IDs are on the right. Below the table, a message indicates that 7 rows were selected. The prompt 'SQL>' is followed by a cursor.

```
SQL> select name, stu_id
2  from student
3  where year = 3 or year = 4
4  order by name, stu_id;
```

NAME	STU_ID
김관식	1292001
김우주	1292305
김정현	1292002
김정현	1292003
김광수	1292303
박철수	1292501
백태성	1292502

7 개의 행이 선택되었습니다.

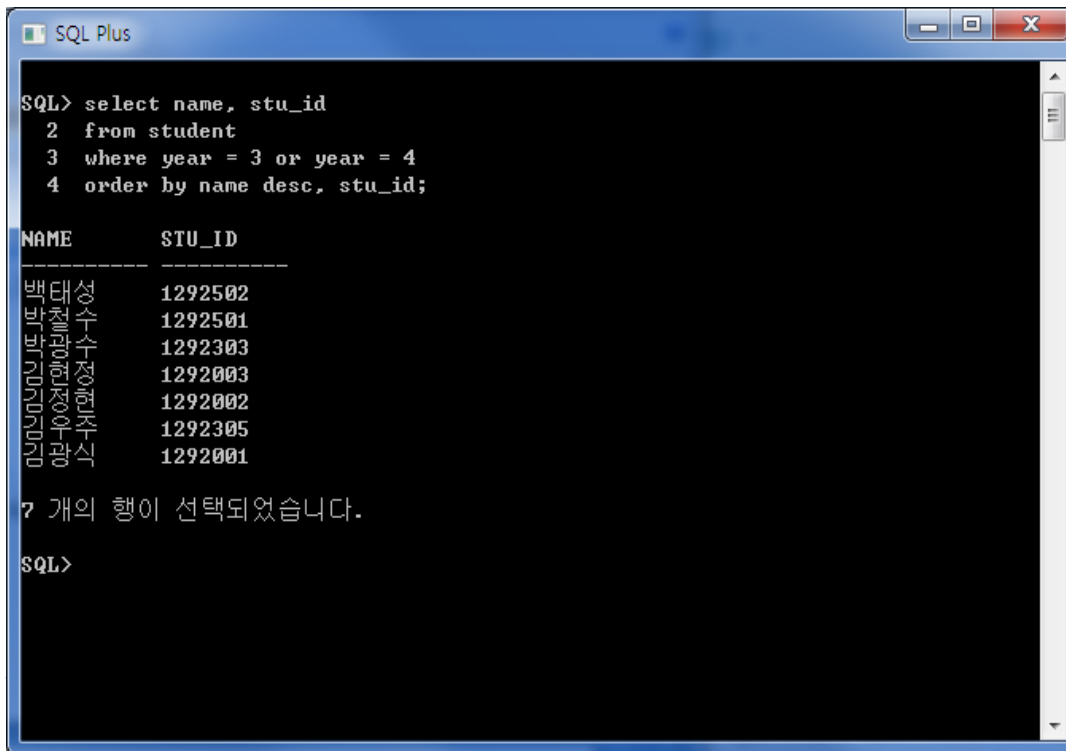
```
SQL> _
```

레코드의 순서 지정

- ▶ 내림차순은 해당 필드 이름 뒤에 **desc** 라는 키워드를 삽입

(질의 28)

```
select    name, stu_id
from      student
where     year = 3 or year = 4
order by  name desc, stu_id
```



```
SQL> select name, stu_id
2  from student
3  where year = 3 or year = 4
4  order by name desc, stu_id;
```

NAME	STU_ID
백태성	1292502
박철수	1292501
박정수	1292303
김민준	1292003
김민준	1292002
김민준	1292305
김광식	1292001

7 개의 행이 선택되었습니다.

```
SQL>
```

재명명 연산

- ▶ 테이블이나 필드에 대한 재명명
 - ▶ 실제 테이블 이름이 수정되거나 필드 이름이 바뀌는 것이 아님
 - ▶ 질의를 처리하는 과정 동안만 일시적으로 사용
 - ▶ 표현이 단순화하거나, 동일 이름이 존재할 경우에 사용
- ▶ 예) student 테이블과 department 테이블을 조인하여 학생들의 이름과 소속 학과 이름을 검색

(질의 29)

```
select  student.name, department.dept_name
from    student, department
where   student.dept_id = department.dept_id
```



(질의 30)

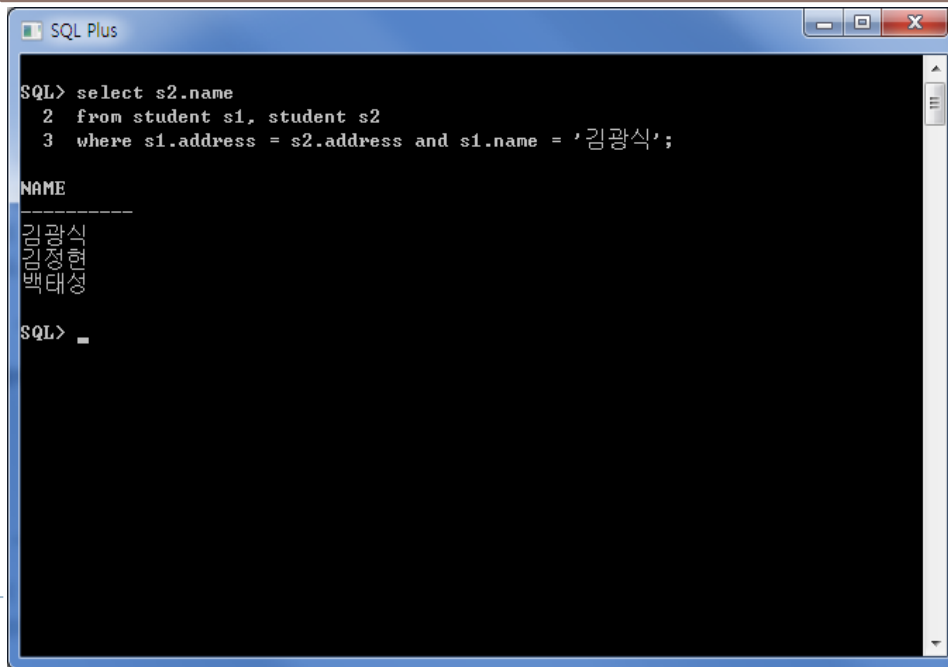
```
select  s.name, d.dept_name
from    student s, department d
where   s.dept_id = d.dept_id
```

재명명 연산

- ▶ 동일 테이블이 두 번 사용되는 예
- ▶ 예) student 테이블에서 '김광식' 학생과 주소가 같은 학생들의 이름과 주소를 검색

(질의 31)

```
select    s2.name
from      student s1, student s2
where      s1.address = s2.address and s1.name = '김광식'
```



The screenshot shows a SQL Plus window with a blue title bar. The command window contains the following text:

```
SQL> select s2.name
2  from student s1, student s2
3  where s1.address = s2.address and s1.name = '김광식';
```

The results window shows the output:

```
NAME
-----
김광식
김광식
김광식
```

The prompt 'SQL>' is visible at the bottom of the command window.

필드의 재명명

- ▶ 질의 실행 결과를 출력할 때 원래 필드의 이름 대신 재명명된 이름으로 출력시키고자 할 때 사용
- ▶ 예) 교수들의 이름과 직위, 재직연수를 출력

(질의 32)

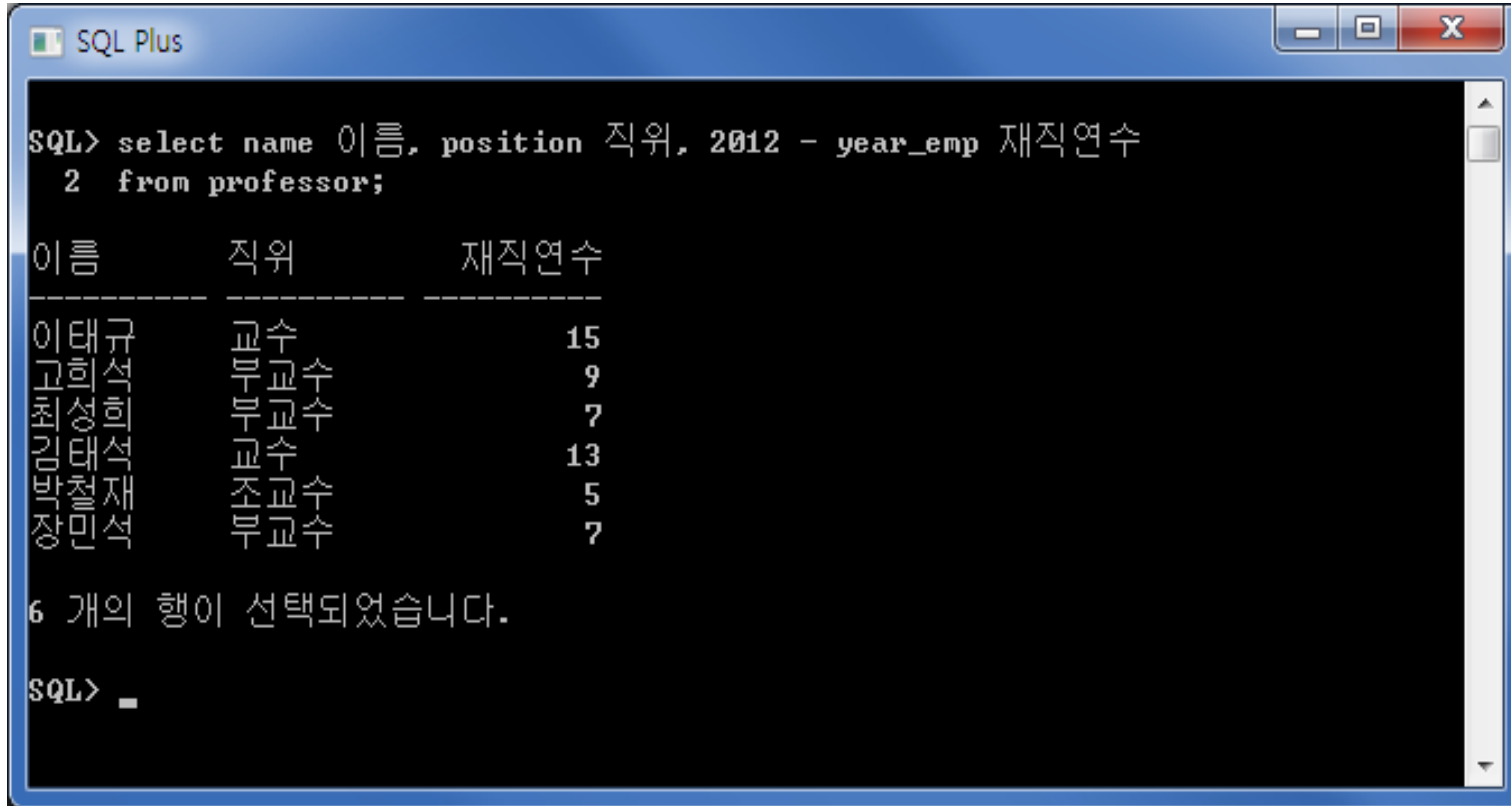
```
select name, position, 2012-year_emp  
from professor
```



(질의 33)

```
select name 이름, position 직위, 2012-year_emp 재직연수  
from professor
```


필드의 재명명



The screenshot shows a SQL Plus window with a query executed. The query uses column aliases in Korean: '이름' for name, '직위' for position, and '재직연수' for years of service. The results are displayed in a table format with dashed lines as separators.

```
SQL> select name 이름, position 직위, 2012 - year_emp 재직연수
2  from professor;
```

이름	직위	재직연수
이태규	교수	15
고희석	부교수	9
최성희	부교수	7
김태석	교수	13
박철재	조교수	5
장민석	부교수	7

6 개의 행이 선택되었습니다.

```
SQL> _
```

LIKE 연산자

- ▶ 문자열에 대해서는 일부분만 일치하는 경우를 찾아야 할 때 사용
- ▶ '=' 연산자 대신에 'like' 연산자를 이용함
 - ▶ '='는 정확히 일치하는 경우에만 사용
- ▶ 형식

where <필드이름> **like** <문자열패턴>

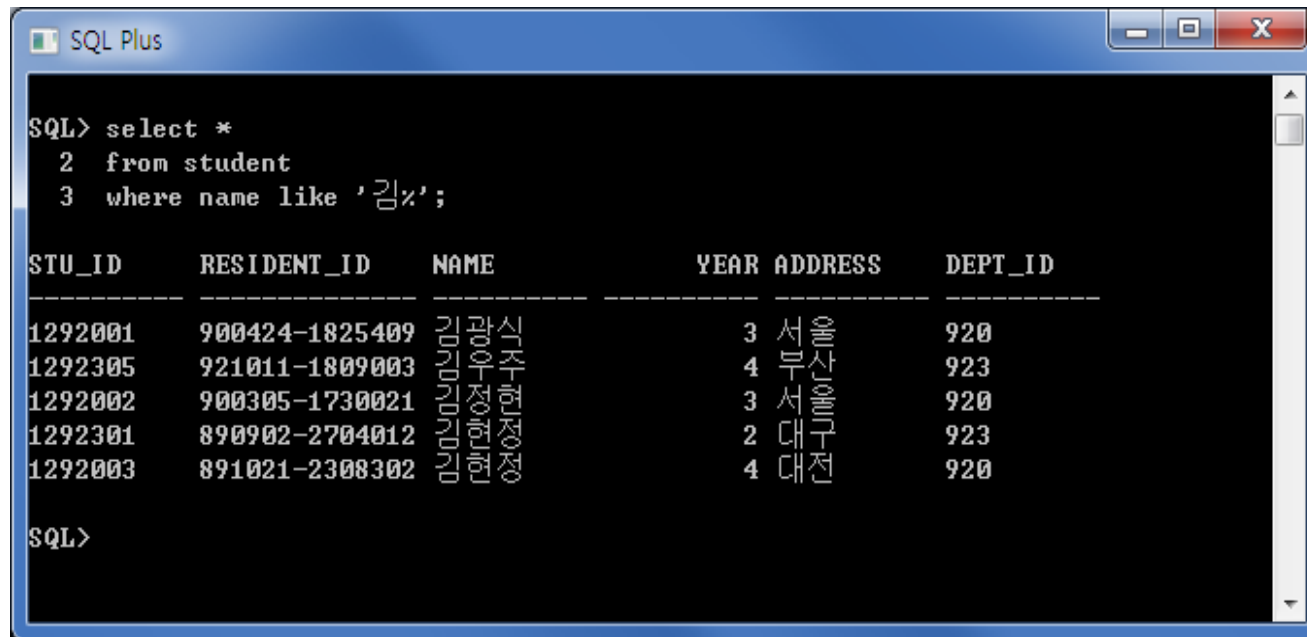
 - ▶ <필드이름>에 지정된 <문자열패턴>이 들어 있는지를 판단
- ▶ 문자열 패턴 종류
 - ▶ _ : 임의의 한 개 문자를 의미한다
 - ▶ % : 임의의 여러 개 문자를 의미한다
 - ▶ 예)
 - '%서울%' : '서울'이란 단어가 포함된 문자열
 - '%서울' : '서울'이란 단어로 끝나는 문자열
 - '서울%' : '서울'이란 단어로 시작하는 문자열
 - '___' : 정확히 세 개의 문자로 구성된 문자열
 - '___%' : 최소한 세 개의 문자로 구성된 문자열

LIKE 연산자

- ▶ student 테이블에서 김씨 성을 가진 학생들을 찾는 질의

(질의 34)

```
select *  
from student  
where name like '김%'
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select *  
  2  from student  
  3  where name like '김%';
```

STU_ID	RESIDENT_ID	NAME	YEAR	ADDRESS	DEPT_ID
1292001	900424-1825409	김광식	3	서울	920
1292305	921011-1809003	김우주	4	부산	923
1292002	900305-1730021	김정현	3	서울	920
1292301	890902-2704012	김현정	2	대구	923
1292003	891021-2308302	김현정	4	대전	920

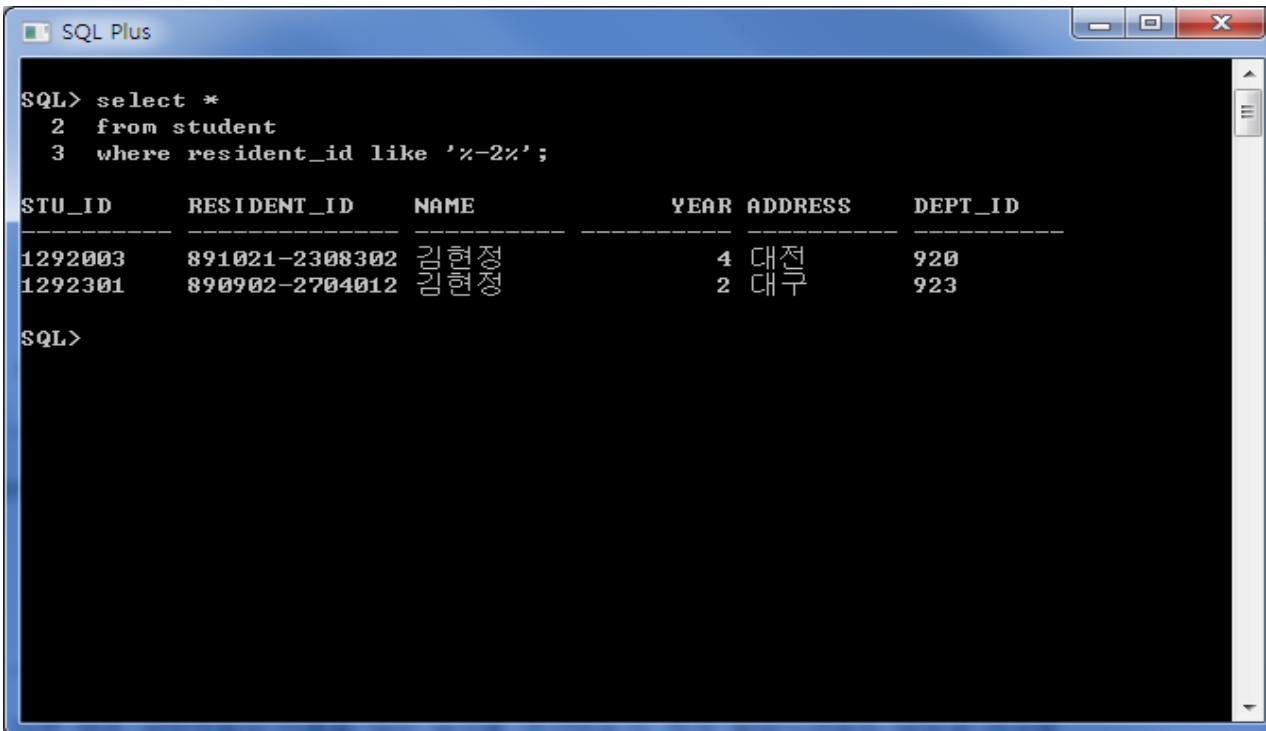
SQL>

LIKE 연산자

- ▶ student 테이블에서 여학생들만을 검색

(질의 35)

```
select *  
from student  
where resident_id like '%-2%'
```



```
SQL> select *  
2  from student  
3  where resident_id like '%-2%';
```

STU_ID	RESIDENT_ID	NAME	YEAR	ADDRESS	DEPT_ID
1292003	891021-2308302	김현정	4	대전	920
1292301	890902-2704012	김현정	2	대구	923

```
SQL>
```

집합연산

- ▶ 관계대수의 집합 연산인 합집합, 교집합, 차집합에 해당하는 연산자
 - ▶ union
 - ▶ intersect
 - ▶ minus
- ▶ 형식

`<select문 1> <집합연산자> <select문 2>`
- ▶ 조건
 - ▶ <select문 1> 과 <select문 2>의 필드의 개수와 데이터타입이 서로 같아야 함

UNION

- ▶ 예) student 테이블의 학생 이름과 professor 테이블의 교수 이름을 합쳐서 출력

(질의 36)

```
select name from student
union
select name from professor
```

```

SQL> select name from student
      2 union
      3 select name from professor;

NAME
-----
고희석
고광주
김정현
김태석
김현정
김수주
박광수
박철수
박재성
백태성
이태규

NAME
-----
장민석
최성희
홍길동

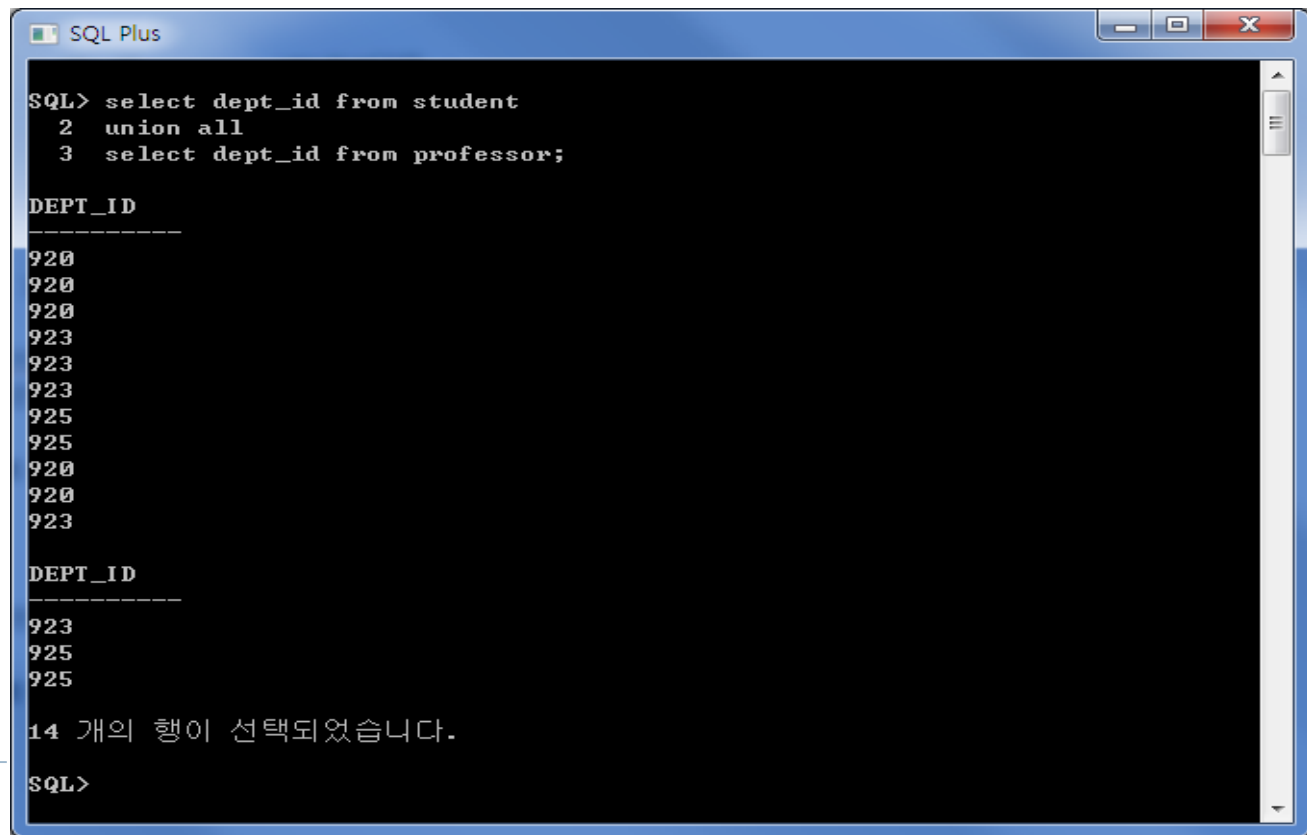
14 개의 행이 선택되었습니다.

SQL>

```

UNION ALL

- ▶ **union** 연산자는 연산 결과에 중복되는 값이 들어갈 경우 한번만
- ▶ 중복을 제거하고 싶지 않다면 **union** 연산자 대신 **union all** 연산자를 사용한다.
- ▶ 예) student 테이블과 professor 테이블에서 학과번호를 중복을 허용하여 출력



```
SQL> select dept_id from student
2 union all
3 select dept_id from professor;

DEPT_ID
-----
920
920
920
923
923
923
925
925
920
920
923

DEPT_ID
-----
923
925
925

14 개의 행이 선택되었습니다.

SQL>
```

INTERSECT

- ▶ 예) 컴퓨터공학과 학생들 중에서 교과목에 상관없이 학점을 'A+' 받은 학생들의 학번을 검색

(질의 37)

```
select distinct s.stu_id
from student s, department d, takes t
where s.dept_id = d.dept_id and
      t.stu_id = s.stu_id and
      dept_name='컴퓨터공학과' and grade = 'A+'
```

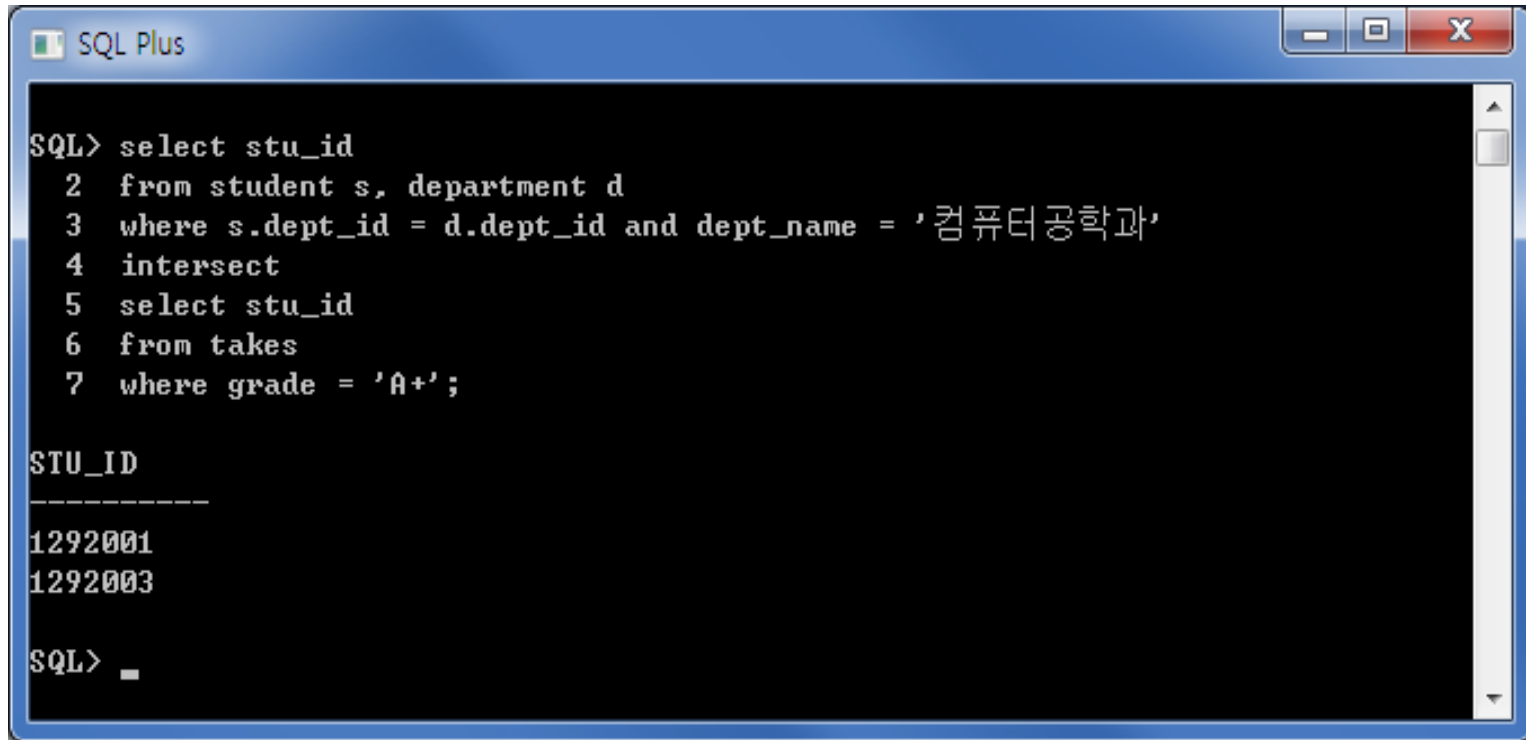
- ▶ 발상의 전환

- ▶ '컴퓨터공학과'에 다니는 학생들의 학번과 takes 테이블에서 학점이 'A+'인 학생들의 학번의 교집합

(질의 40)

```
select stu_id
from student s, department d
where s.dept_id = d.dept_id and dept_name='컴퓨터공학과'
intersect
select stu_id
from takes
where grade = 'A+';
```


INTERSECT



The screenshot shows a window titled "SQL Plus" with a black background and white text. The text displays an SQL query using the INTERSECT operator to find student IDs who are in the '컴퓨터공학과' (Computer Engineering) department and have an 'A+' grade in their courses. The results show two student IDs: 1292001 and 1292003.

```
SQL> select stu_id
  2  from student s, department d
  3  where s.dept_id = d.dept_id and dept_name = '컴퓨터공학과'
  4  intersect
  5  select stu_id
  6  from takes
  7  where grade = 'A+';
```

STU_ID

1292001
1292003

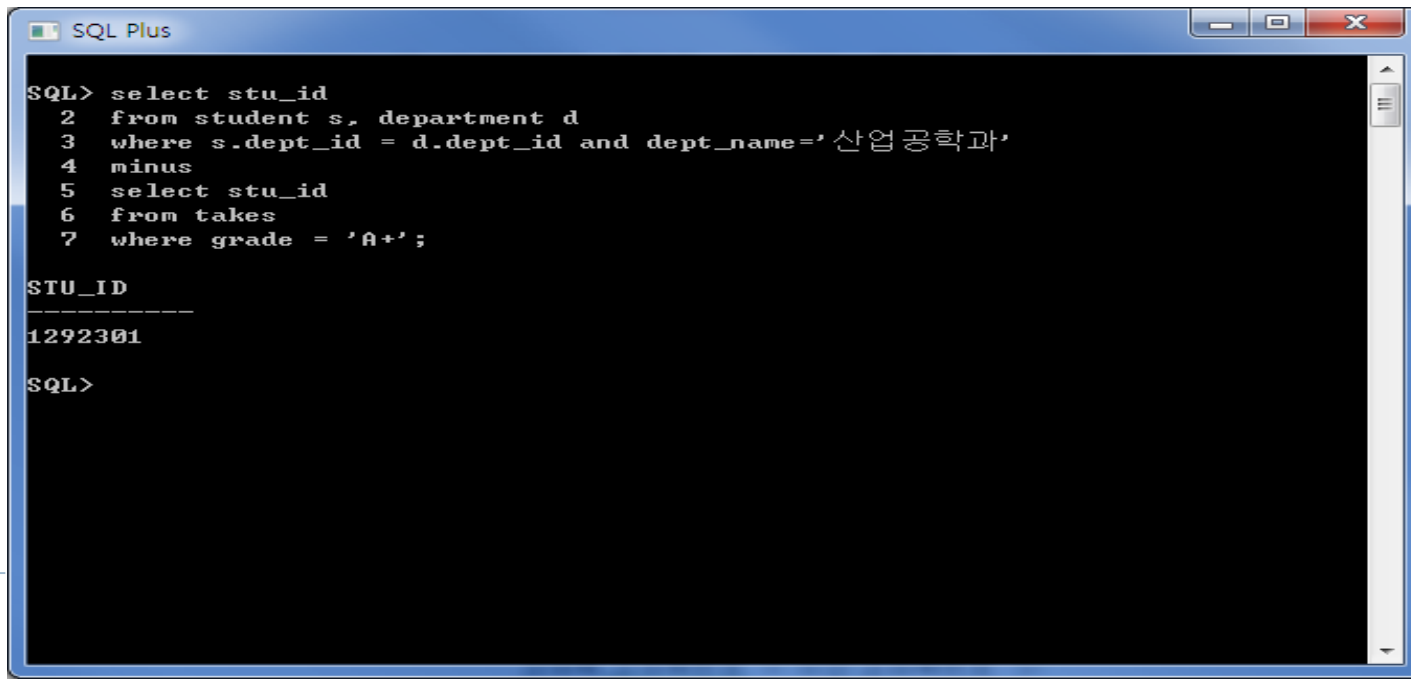
SQL> _

MINUS

- ▶ 예) 산업공학과 학생들 중에서 한번이라도 'A+'를 받지 못한 학생들의 학번을 검색

(질의 41)

```
select stu_id from student s, department d
where s.dept_id = d.dept_id and dept_name='산업공학과'
minus
select stu_id from takes
where grade = 'A+'
```

A screenshot of a SQL Plus window titled "SQL Plus". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main area is black with white text. The SQL prompt "SQL>" is followed by a query using the MINUS operator. The query selects student IDs from the 'student' and 'department' tables where the department name is '산업공학과', and then subtracts the student IDs from the 'takes' table where the grade is 'A+'. The result shows a single student ID, 1292301, under the column heading 'STU_ID'.

```
SQL> select stu_id
  2  from student s, department d
  3  where s.dept_id = d.dept_id and dept_name='산업공학과'
  4  minus
  5  select stu_id
  6  from takes
  7  where grade = 'A+' ;

STU_ID
-----
1292301

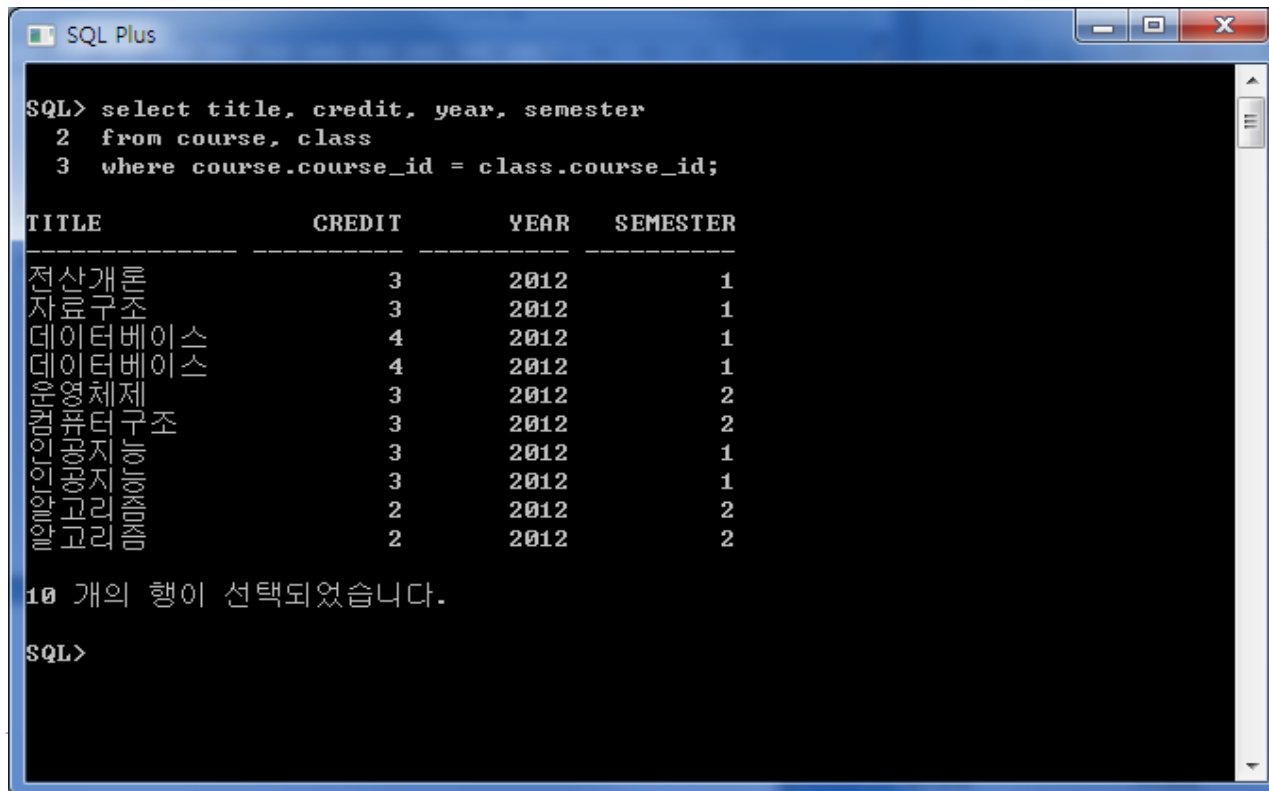
SQL>
```

외부조인(outer join)

- ▶ 예) 모든 교과목들에 대해 교과목명, 학점수, 개설 년도, 개설 학기를 검색

(질의 42)

```
select    title, credit, year, semester
from      course, class
where     course.course_id = class.course_id
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select title, credit, year, semester
2  from course, class
3  where course.course_id = class.course_id;
```

TITLE	CREDIT	YEAR	SEMESTER
전산개론	3	2012	1
자료구조	3	2012	1
데이터베이스	4	2012	1
데이터베이스	4	2012	1
운영체제	3	2012	2
컴퓨터구조	3	2012	2
공인영어	3	2012	1
공인영어	3	2012	1
고리	2	2012	2
고리	2	2012	2

10 개의 행이 선택되었습니다.

```
SQL>
```

외부조인(outer join)

- ▶ 강좌로 개설된 적이 있는 교과목에 대해서만 검색됨
 - ▶ '이산수학', '객체지향언어' 교과목들은 class 테이블에 저장되어 있지 않기 때문에 검색 결과에 포함되지 못함

왼쪽 외부조인(left outer join)

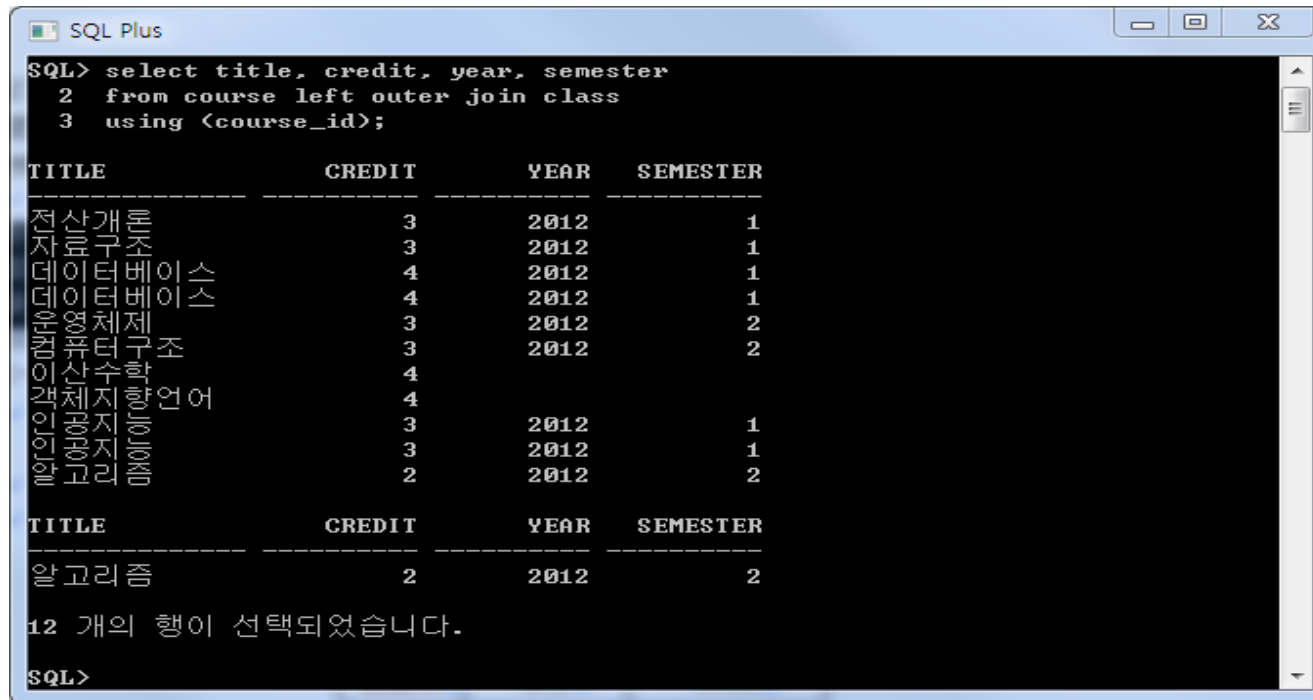
- ▶ 연산자의 왼쪽에 위치한 테이블의 각 레코드에 대해서 오른쪽 테이블에 조인 조건에 부합하는 레코드가 없을 경우에도 검색 결과에 포함
- ▶ 생성되는 결과 레코드에서 오른쪽 테이블의 나머지 필드에는 널이 삽입

(질의 43)

```
select title, credit, year, semester  
from course left outer join class  
using (course_id)
```

- ▶ 'course **left outer join** class'
 - ▶ course 테이블과 class 테이블에 대해 왼쪽 외부조인을 적용
- ▶ '**using** (course_id)'
 - ▶ 조인 조건이 'course.course_id = class.course_id'라는 것을 의미

왼쪽 외부조인(left outer join)



```
SQL> select title, credit, year, semester
2   from course left outer join class
3   using (course_id);
```

TITLE	CREDIT	YEAR	SEMESTER
전산개론	3	2012	1
자료구조	3	2012	1
데이터베이스	4	2012	1
데이터베이스	4	2012	1
운영체제	3	2012	2
컴퓨터구조	3	2012	2
이산수학	4		
객체지향언어	4		
인공지능	3	2012	1
인공지능	3	2012	1
알고리즘	2	2012	2

12 개의 행이 선택되었습니다.

```
SQL>
```

▶ 동일한 표현

(질의 44)

select	title, credit, year, semester
from	course, class
where	course.course_id = class.course_id (+)

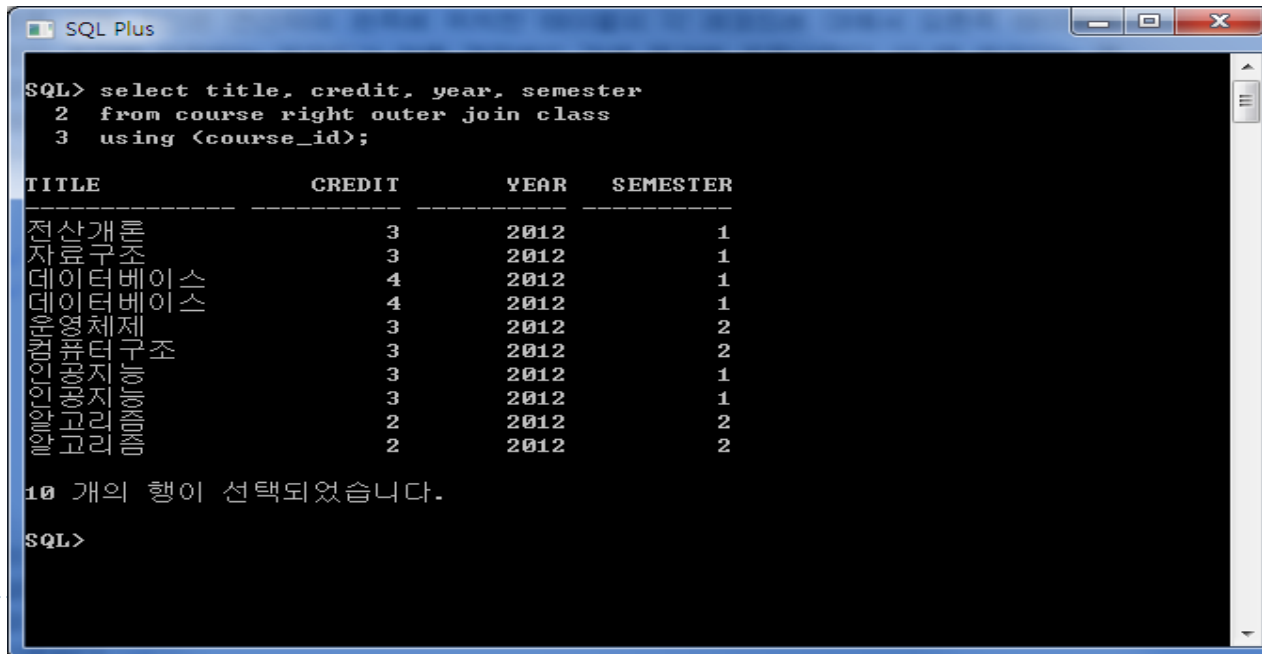
오른쪽 외부조인(right outer join)

(질의 45)

```
select title, credit, year, semester
from course right outer join class
using (course_id)
```

(질의 46)

```
select title, credit, year, semester
from course, class
where course.course_id (+) = class.course_id
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select title, credit, year, semester
2  from course right outer join class
3  using (course_id);
```

TITLE	CREDIT	YEAR	SEMESTER
전산개론	3	2012	1
자료구조	3	2012	1
데이터베이스	4	2012	1
데이터베이스	4	2012	1
운영체제	3	2012	2
컴퓨터구조	3	2012	2
컴퓨터지능	3	2012	1
컴퓨터지능	3	2012	1
고급리눅스	2	2012	2
고급리눅스	2	2012	2

10 개의 행이 선택되었습니다.

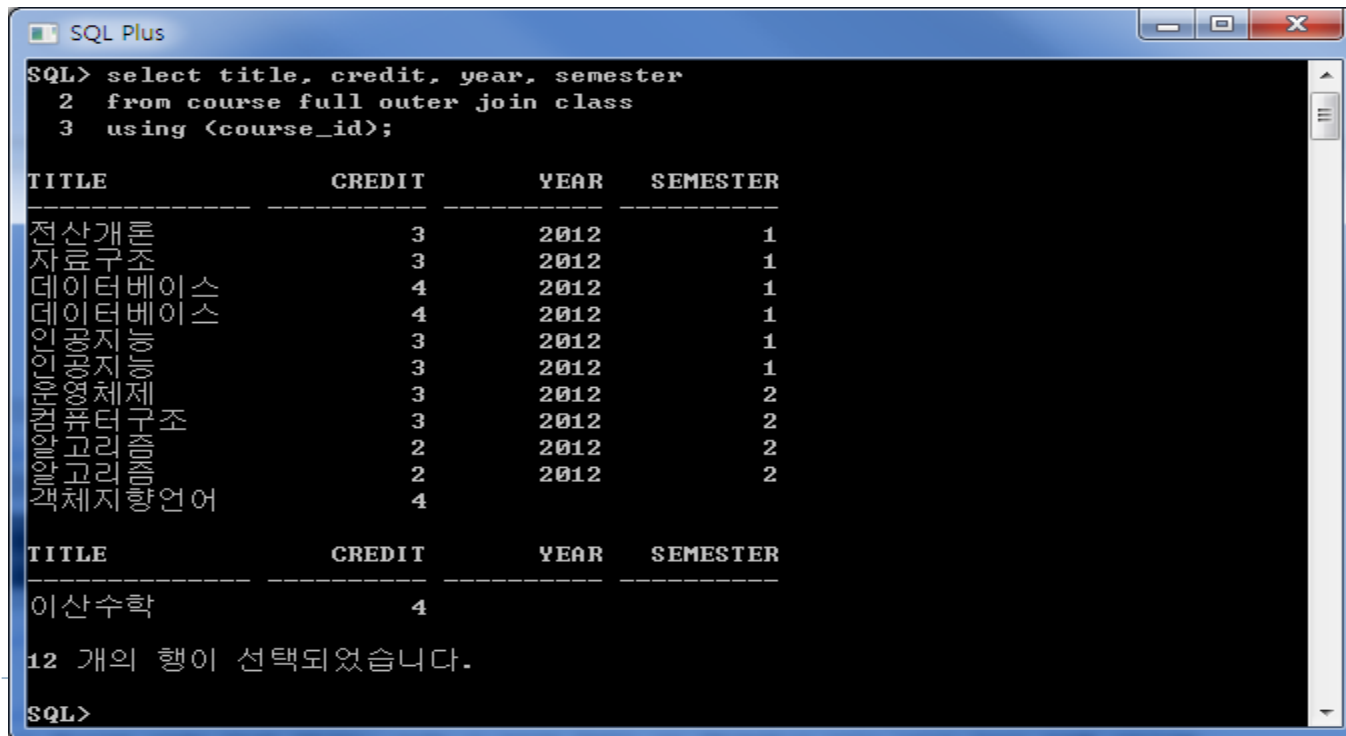
```
SQL>
```

완전 외부조인(full outer join)

- ▶ 양쪽 테이블에서 서로 일치하는 레코드가 없을 경우, 해당 레코드들도 결과 테이블에 포함시키며 나머지 필드에 대해서는 모두 널을 삽입

(질의 47)

```
select title, credit, year, semester
from course full outer join class
using (course_id)
```



SQL Plus window showing the execution of a full outer join query. The query is: `SQL> select title, credit, year, semester
2 from course full outer join class
3 using (course_id);`

The result is displayed in two tables. The first table contains 11 rows of data, and the second table contains 1 row of data. The total number of rows is 12.

TITLE	CREDIT	YEAR	SEMESTER
전산개론	3	2012	1
자료구조	3	2012	1
데이터베이스	4	2012	1
데이터베이스	4	2012	1
이공지능공학	3	2012	1
이공지능공학	3	2012	1
컴퓨터구조	3	2012	2
컴퓨터구조	3	2012	2
고리집합	2	2012	2
고리집합	2	2012	2
객체지향언어	4		

TITLE	CREDIT	YEAR	SEMESTER
이산수학	4		

12 개의 행이 선택되었습니다.

SQL>

집계 함수(aggregate function)

- ▶ 통계연산 기능 제공
- ▶ 예)
 - ▶ 컴퓨터공학과 학생들은 모두 몇 명인가?
 - ▶ 교수들의 평균 재직연수는 몇 년인가?
- ▶ 종류
 - ▶ **count** : 데이터의 개수를 구한다.
 - ▶ **sum** : 데이터의 합을 구한다.
 - ▶ **avg** : 데이터의 평균 값을 구한다.
 - ▶ **max** : 데이터의 최대 값을 구한다.
 - ▶ **min** : 데이터의 최소 값을 구한다.
- ▶ SELECT 절과 HAVING절(뒤에 설명)에서만 사용가능
- ▶ sum, avg는 숫자형 데이터 타입을 갖는 필드에만 적용가능

COUNT

▶ 형식

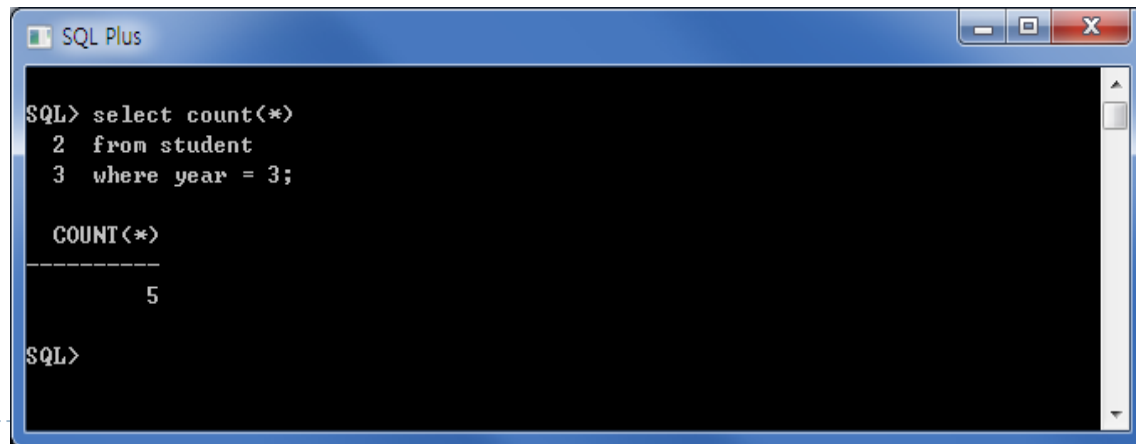
count(distinct <필드이름>)

- ▶ 해당 필드에 값이 몇 개인지 출력
- ▶ distinct: 서로 구별되는 값의 개수가 필요한 경우에만 사용
- ▶ NULL은 계산에서 제외됨
- ▶ 단, <필드이름>에는 필드 이름 대신 '*'가 사용된 경우에는 레코드의 개수를 계산

▶ 예) student 테이블에서 3학년 학생이 몇 명인지 출력

(질의 48)

```
select count(*)  
from student  
where year = 3
```



The screenshot shows a SQL Plus window with a blue title bar. The command prompt shows the execution of a SQL query to count the number of students in the 3rd year. The output shows a single row with the value 5.

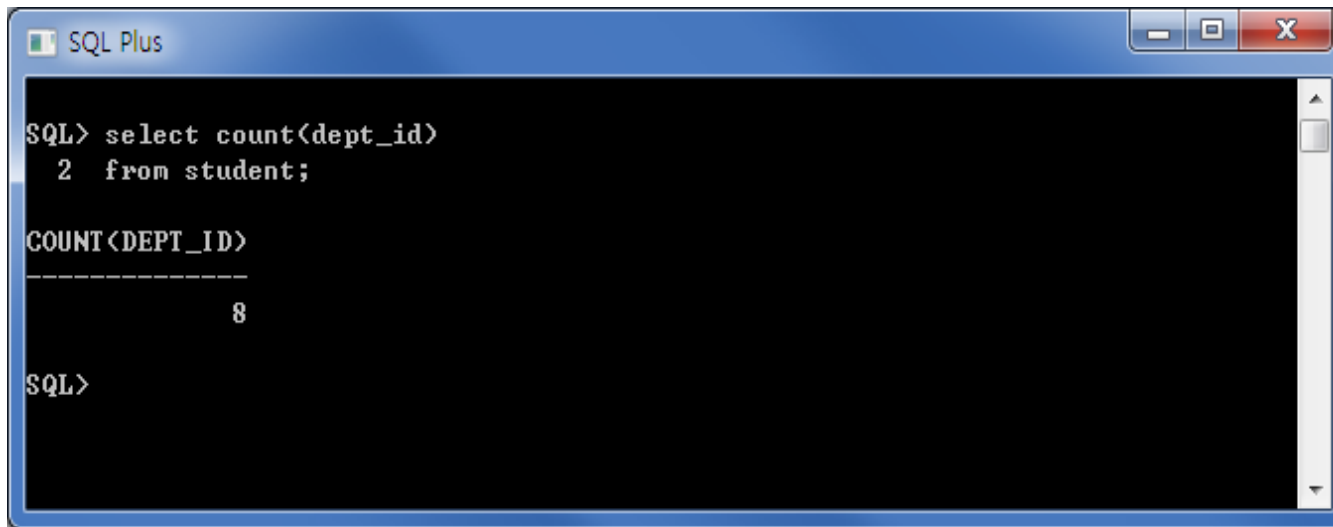
```
SQL Plus  
SQL> select count(*)  
2   from student  
3   where year = 3;  
  
COUNT(*)  
-----  
5  
  
SQL>
```

COUNT

- ▶ 예) student 테이블에서 dept_id 필드에 값이 몇 개인지를 출력

(질의 49)

```
select count(dept_id)
from student
```



The screenshot shows a SQL Plus window with a blue title bar and standard Windows window controls. The main area is black with white text. The prompt 'SQL>' is followed by the query 'select count(dept_id) from student;'. Below the query, the output is displayed: 'COUNT(DEPT_ID)' followed by a horizontal line and the value '8'. The prompt 'SQL>' is shown again at the bottom.

```
SQL> select count(dept_id)
2  from student;

COUNT(DEPT_ID)
-----
                8

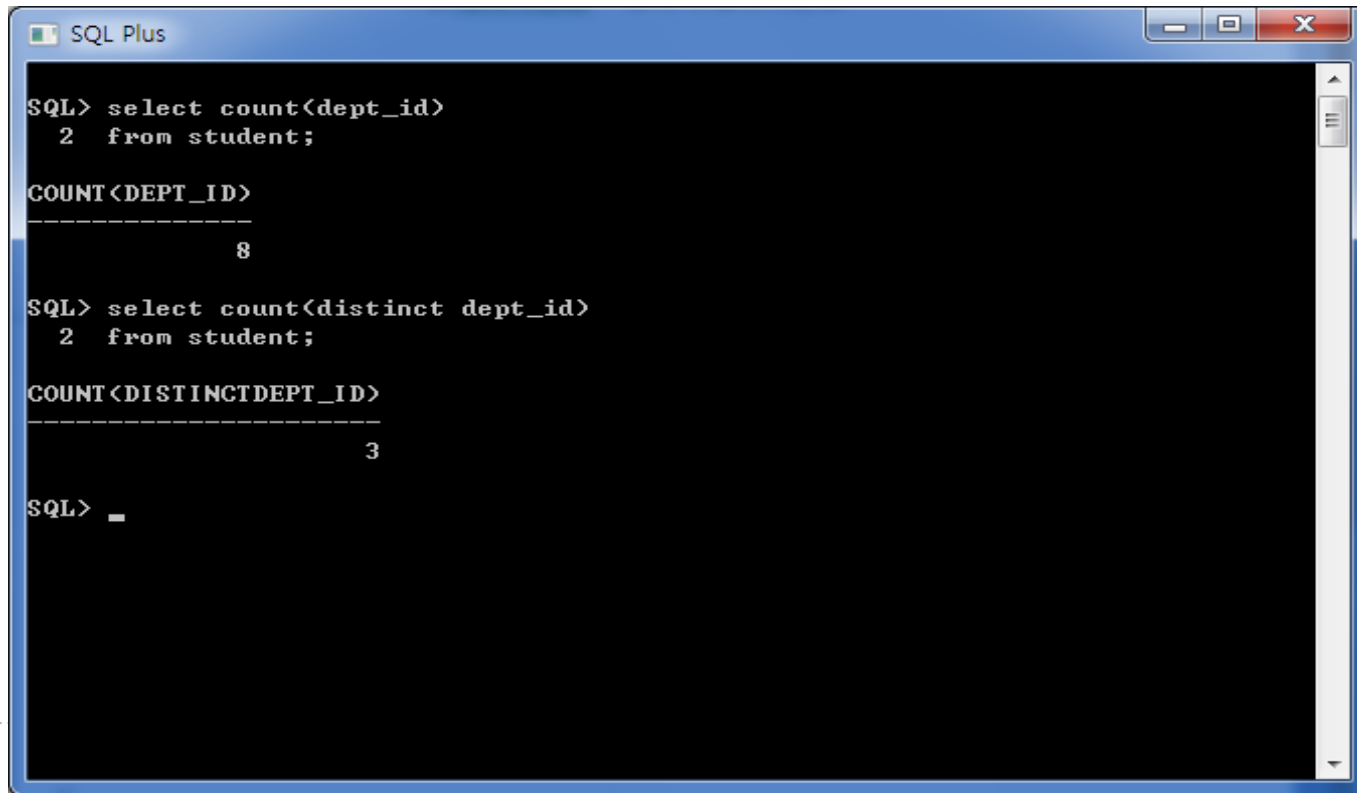
SQL>
```

COUNT

- ▶ distinct 키워드를 사용하면 중복되는 데이터를 제외한 개수를 리턴
- ▶ 예) **count**(dept_id) 대신 **count(distinct dept_id)**를 사용

(질의 49)

```
select count(distinct dept_id)
from student
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select count<dept_id>
2  from student;

COUNT<DEPT_ID>
-----
                8

SQL> select count<distinct dept_id>
2  from student;

COUNT<DISTINCTDEPT_ID>
-----
                3

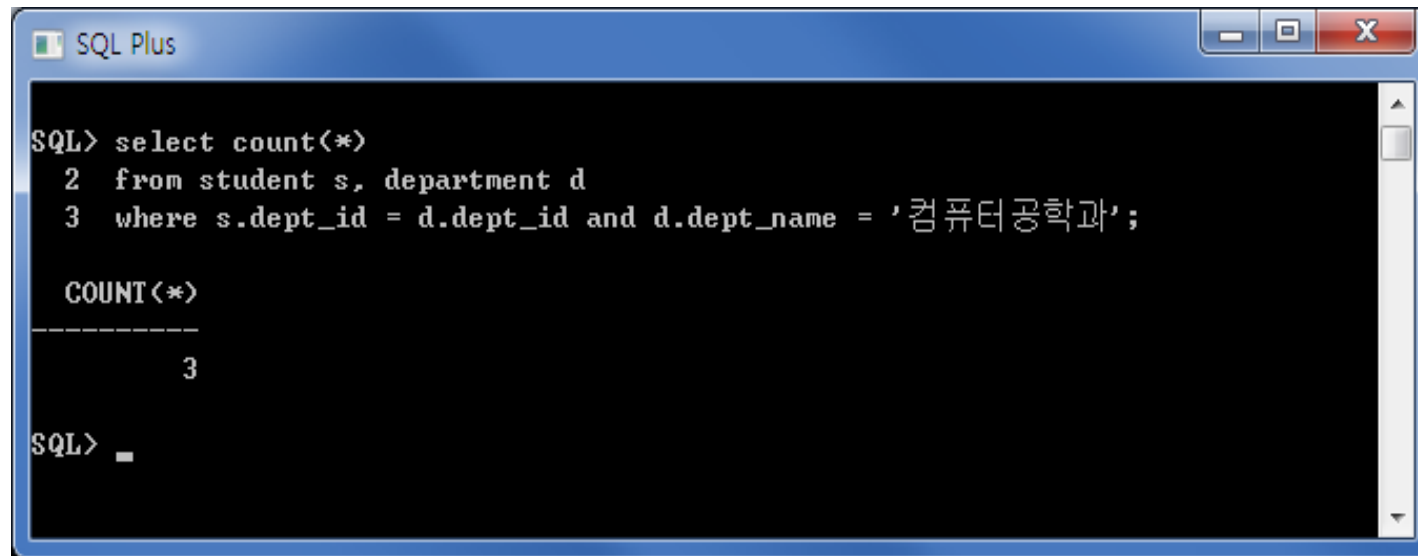
SQL> _
```

COUNT

- ▶ 예) 컴퓨터공학과와 학생 수를 출력

(질의 50)

```
select count(*)  
from student s, department d  
where s.dept_id = d.dept_id and d.dept_name = '컴퓨터공학과'
```



The screenshot shows a SQL Plus window with a blue title bar. The command prompt displays the following SQL query and its result:

```
SQL> select count(*)  
 2  from student s, department d  
 3  where s.dept_id = d.dept_id and d.dept_name = '컴퓨터공학과';  
  
COUNT(*)  
-----  
          3  
  
SQL> _
```

SUM

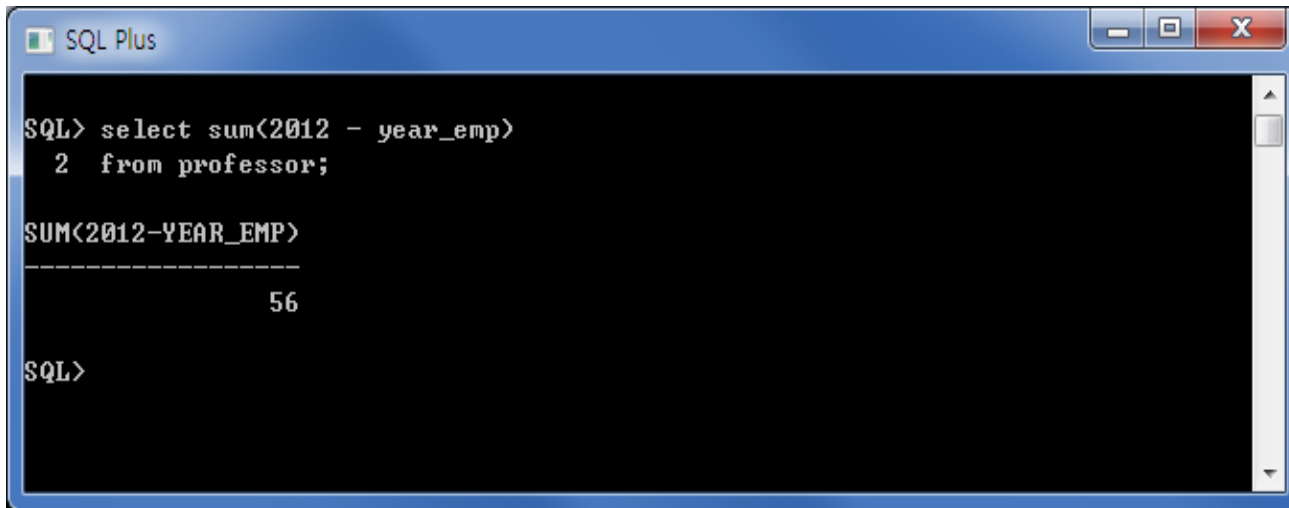
▶ 형식

sum(<필드이름>)

▶ 예) 전체 교수들의 재직연수 합

(질의 51)

```
select    sum(2012 - year_emp)
from      professor
```



The screenshot shows a SQL Plus window with a blue title bar. The command prompt shows the execution of the query: `SQL> select sum(2012 - year_emp)` followed by a blank line and `2 from professor;`. The output displays the column header `SUM(2012-YEAR_EMP)` followed by a horizontal line and the value `56`. The prompt `SQL>` is visible at the bottom.

```
SQL> select sum(2012 - year_emp)
      2  from professor;

SUM(2012-YEAR_EMP)
-----
                56

SQL>
```

SUM

▶ 예)

emp

필드 이름	설명
EMPNO	사원번호
ENAME	사원이름
JOB	업무
MGR	관리자번호
HIREDATE	입사날짜
SAL	급여
COMM	커미션
DEPTNO	부서코드

dept

필드 이름	설명
DEPTNO	부서코드
DNAME	부서이름
LOC	위치

SUM

- ▶ emp 테이블에 저장된 모든 직원들의 급여 합을 출력

(질의 52)

```
select  sum(sal)
from    emp
```

- ▶ 업무(job 필드)가 'ANALYST'인 직원들의 급여의 합을 출력

(질의 53)

```
select  sum(sal)
from    emp
where   job = 'ANALYST'
```

- ▶ 부서 이름이 'RESEARCH'인 직원들의 급여의 합을 출력

(질의 54)

```
select  sum(sal)
from    emp e, dept d
where   e.deptno = d.deptno and dname = 'RESEARCH'
```


AVG

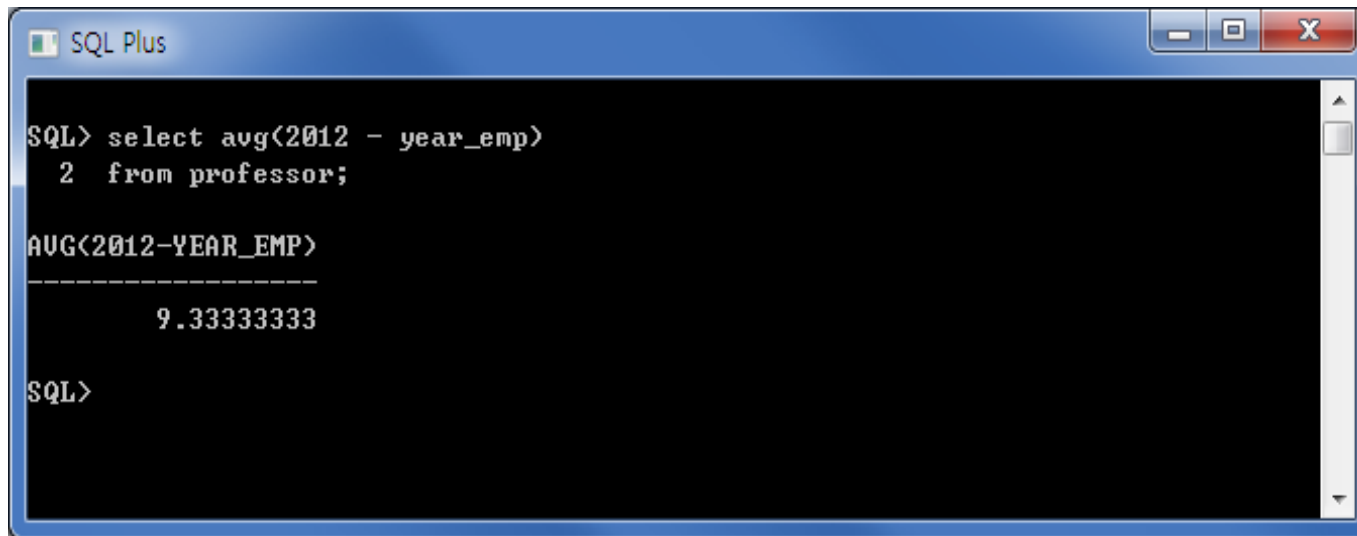
▶ 형식

avg(<필드이름>)

▶ 예) 전체 교수의 평균 재직연수를 출력

(질의 55)

```
select    avg(2012 - year_emp)
from      professor
```



The screenshot shows a window titled "SQL Plus" with a black background and white text. The text displays the execution of an SQL query and its result.

```
SQL> select avg(2012 - year_emp)
      2  from professor;

AUG(2012-YEAR_EMP)
-----
          9.33333333

SQL>
```

MIN, MAX

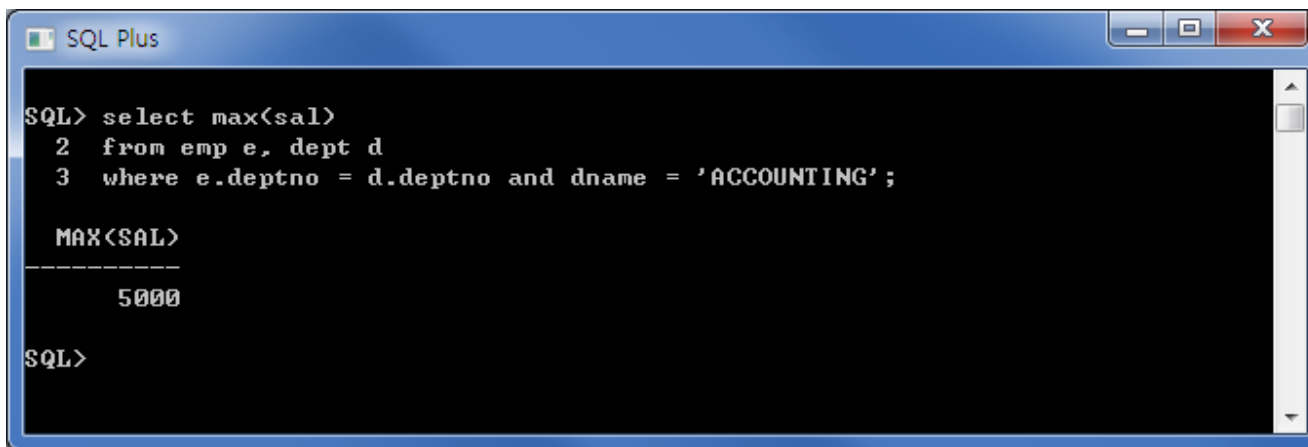
▶ 형식

```
max(<필드이름>)  
min(<필드이름>)
```

▶ 예) 부서 이름이 'ACCOUNTING'인 직원들 중에서 최대 급여가 얼마인지 출력

(질의 56)

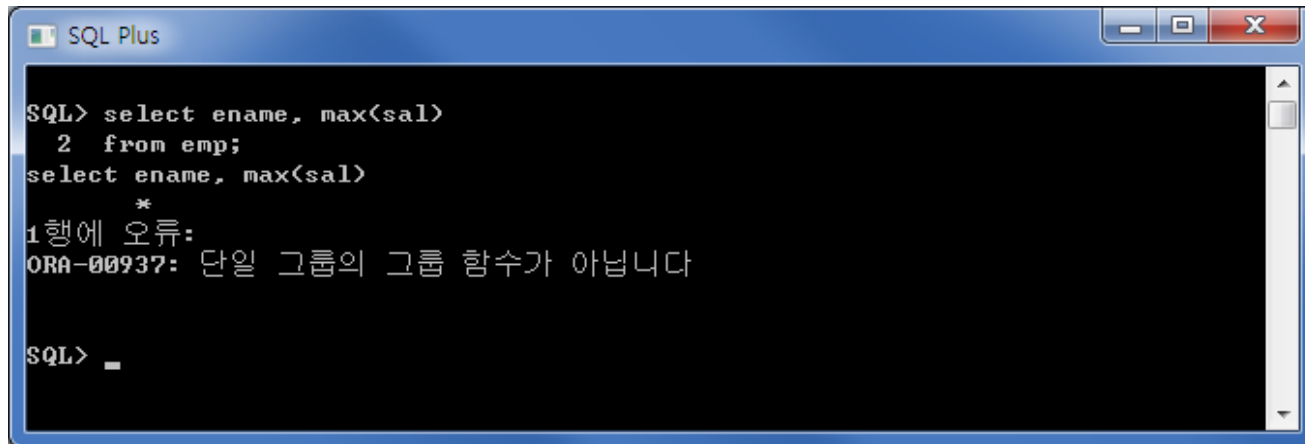
```
select    max(sal)  
from      emp e, dept d  
where     e.deptno = d.deptno and dname = 'ACCOUNTING'
```



```
SQL Plus  
SQL> select max(sal)  
2  from emp e, dept d  
3  where e.deptno = d.deptno and dname = 'ACCOUNTING';  
  
MAX(SAL)  
-----  
      5000  
  
SQL>
```

GROUP BY

- ▶ **select** 절에 집계 함수가 사용될 경우 다른 필드는 **select** 절에 사용할 수가 없음
- ▶ 다음 질의는 오류



```
SQL> select ename, max(sal)
  2   from emp;
select ename, max(sal)
      *
1 행에 오류:
ORA-00937: 단일 그룹의 그룹 함수가 아닙니다

SQL> _
```

- ▶ 지금까지의 SQL은 전체 레코드들을 대상으로 평균, 합, 최대값/최소값만을 출력
- ▶ GROUP BY를 이용하면 그룹별로 집계함수 적용 가능
 - ▶ 예) '학과별 학생 수', '부서별 최대 급여'

GROUP BY

▶ 형식

group by <필드리스트>

- ▶ group by 절은 select문에서 where절 다음에 위치
- ▶ group by 에 지정된 필드의 값이 같은 레코드들끼리 그룹을 지어 각 그룹별로 집계 함수를 적용한 결과를 출력

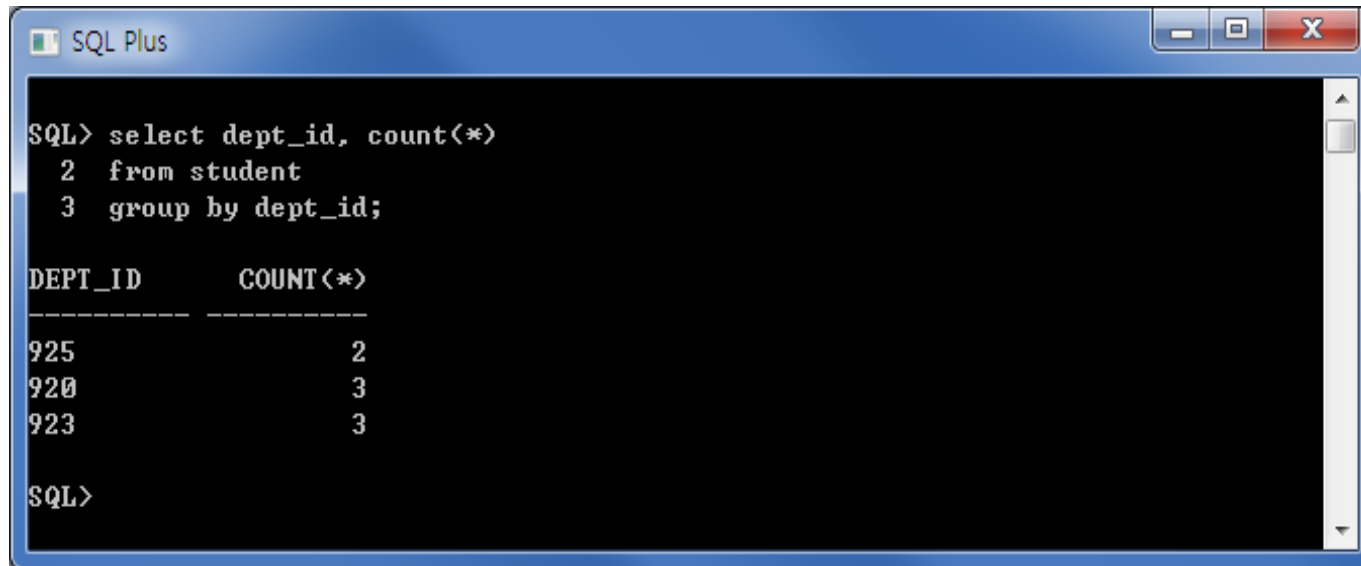
- ▶ 예) student 테이블에서 학과번호(dept_id 필드)별로 레코드의 개수를 출력

(질의 57)

```
select      dept_id, count(*)
from        student
group by    dept_id
```

- ▶ group by 절에 사용된 필드를 select 절에 추가하여 사용할 수 있음

GROUP BY



The screenshot shows a SQL Plus window with a blue title bar. The command prompt shows the execution of a SQL query. The output is a table with two columns: DEPT_ID and COUNT(*). The table has three rows of data.

```
SQL> select dept_id, count(*)
  2  from student
  3  group by dept_id;
```

DEPT_ID	COUNT(*)
925	2
920	3
923	3

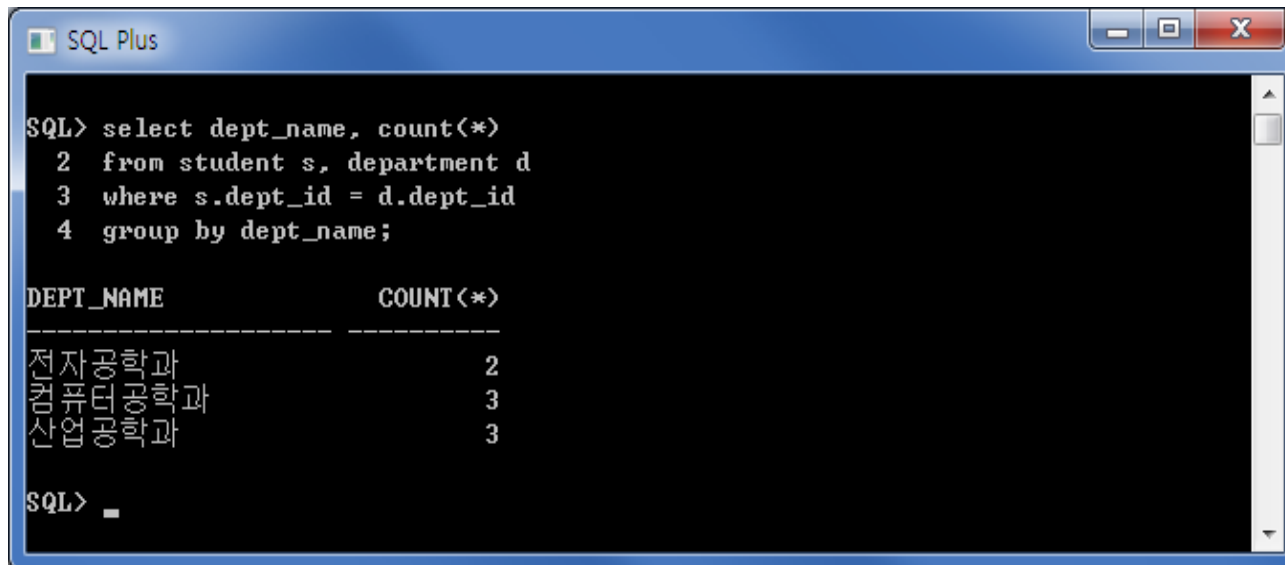
```
SQL>
```

GROUP BY

- ▶ 예) 학과번호 대신 department 테이블과 조인하여 학과 이름이 출력되도록 (질의 57)을 수정

(질의 58)

```
select      dept_name, count(*)
from        student s, department d
where       s.dept_id = d.dept_id
group by    dept_name
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select dept_name, count(*)
2  from student s, department d
3  where s.dept_id = d.dept_id
4  group by dept_name;
```

DEPT_NAME	COUNT(*)
전자공학과	2
컴퓨터공학과	3
산업공학과	3

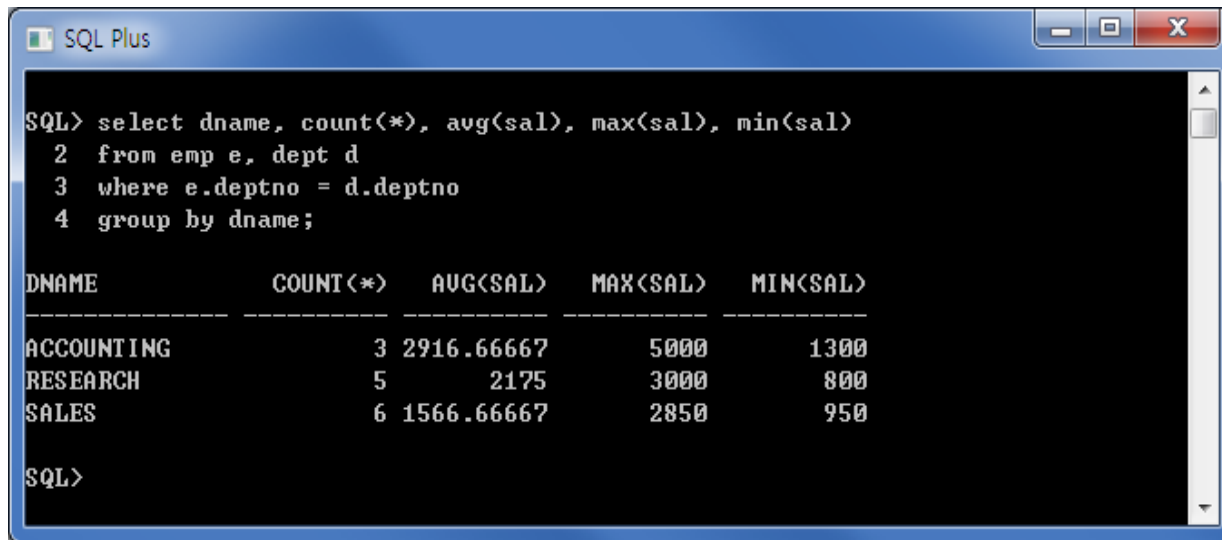
```
SQL> _
```

GROUP BY

- ▶ 예) emp, dept 테이블에서 부서별 직원수, 평균급여, 최대급여, 최소급여를 출력

(질의 59)

```
select      dname, count(*), avg(sal), max(sal), min(sal)
from        emp e, dept d
where       e.deptno = d.deptno
group by    dname
```



The screenshot shows a SQL Plus window with the following SQL query and its results:

```
SQL> select dname, count(*), avg(sal), max(sal), min(sal)
2  from emp e, dept d
3  where e.deptno = d.deptno
4  group by dname;
```

DNAME	COUNT(*)	AUG(SAL)	MAX(SAL)	MIN(SAL)
ACCOUNTING	3	2916.66667	5000	1300
RESEARCH	5	2175	3000	800
SALES	6	1566.66667	2850	950

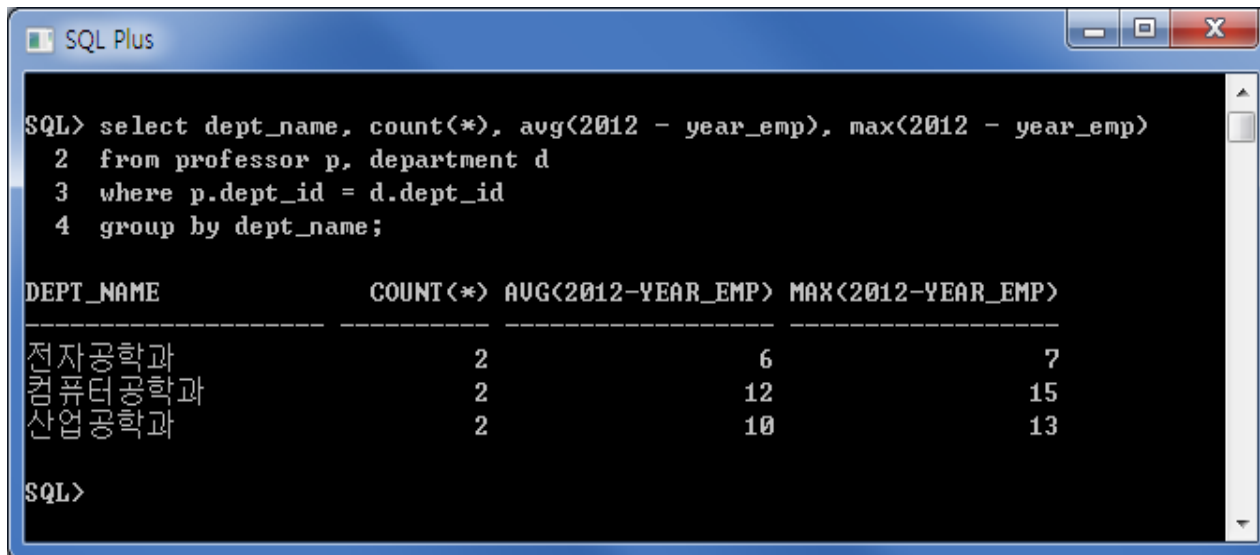
SQL>

GROUP BY

- ▶ 예) 학사 데이터베이스에서 학과별 교수 숫자와 평균 재직연수, 최대 재직연수를 출력

(질의 60)

```
select      dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
from        professor p, department d
where       p.dept_id = d.dept_id
group by    dept_name
```



The screenshot shows a SQL Plus window with the following SQL query and its results:

```
SQL> select dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
2  from professor p, department d
3  where p.dept_id = d.dept_id
4  group by dept_name;
```

DEPT_NAME	COUNT(*)	AUG(2012-YEAR_EMP)	MAX(2012-YEAR_EMP)
전자공학과	2	6	7
컴퓨터공학과	2	12	15
산업공학과	2	10	13

SQL>

HAVING

- ▶ 그룹에 대한 조건을 명시할 때 사용
- ▶ 예) 평균 재직연수가 10년 이상인 학과에 대해서만 교수 숫자와 평균 재직연수, 최대 재직연수를 출력
- ▶ 다음은 오류
 - ▶ Group에 대한 조건은 where절에 사용하지 못함
 - ▶ HAVING절을 이용해야 함

(질의 61)

```
select      dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
from        professor p, department d
where       p.dept_id = d.dept_id and avg(2012 - year_emp) >= 10
group by    dept_name
```

- ▶ 형식

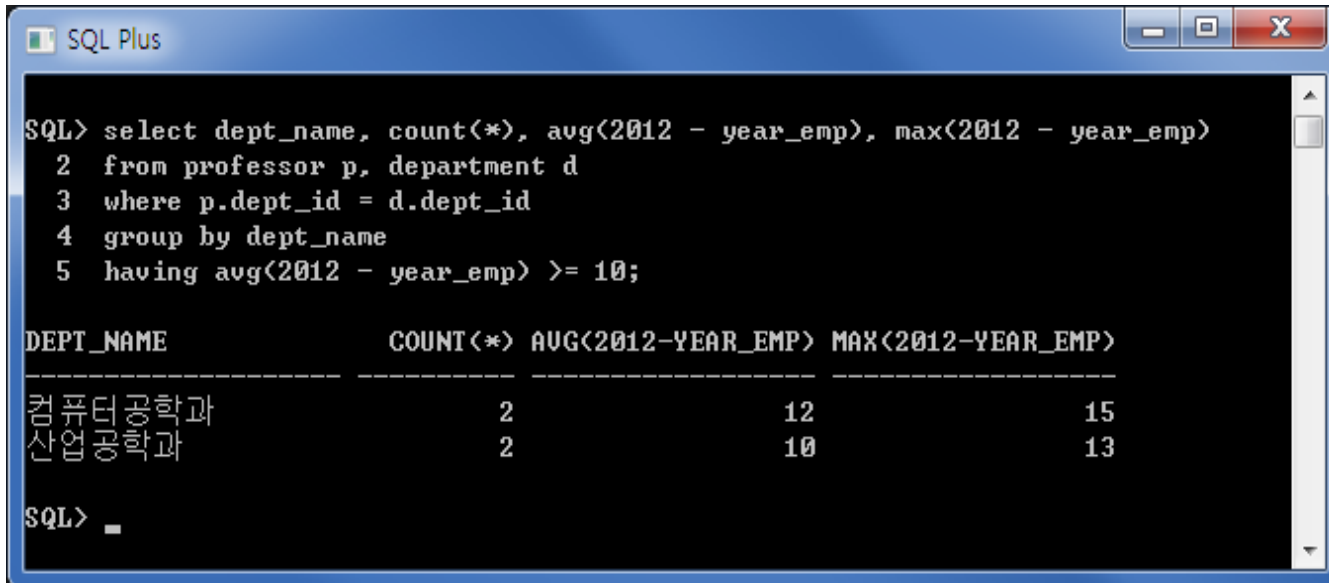
```
having <집계함수 조건>
```

HAVING

- ▶ 예) having 절을 이용하여 다시 작성

(질의 62)

```
select      dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
from        professor p, department d
where       p.dept_id = d.dept_id
group by    dept_name
having      avg(2012 - year_emp) >= 10
```



```
SQL> select dept_name, count(*), avg(2012 - year_emp), max(2012 - year_emp)
  2  from professor p, department d
  3  where p.dept_id = d.dept_id
  4  group by dept_name
  5  having avg(2012 - year_emp) >= 10;
```

DEPT_NAME	COUNT(*)	AVG(2012-YEAR_EMP)	MAX(2012-YEAR_EMP)
컴퓨터공학과	2	12	15
산업공학과	2	10	13

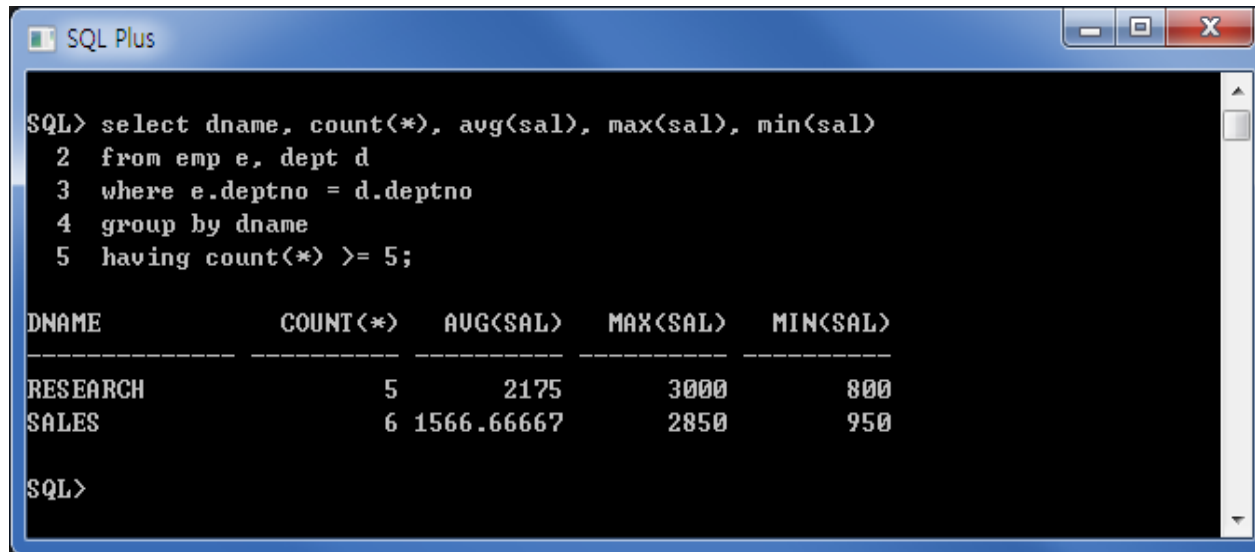
```
SQL> _
```

HAVING

- ▶ 예) 직원 숫자가 5명 이상인 부서에 대해서 부서별 직원수, 평균급여, 최대급여, 최소급여를 출력

(질의 63)

```
select      dname, count(*), avg(sal), max(sal), min(sal)
from        emp e, dept d
where       e.deptno = d.deptno
group by    dname
having      count(*) >= 5
```



The screenshot shows a SQL Plus window with the following content:

```
SQL> select dname, count(*), avg(sal), max(sal), min(sal)
2  from emp e, dept d
3  where e.deptno = d.deptno
4  group by dname
5  having count(*) >= 5;
```

DNAME	COUNT(*)	AUG(SAL)	MAX(SAL)	MIN(SAL)
RESEARCH	5	2175	3000	800
SALES	6	1566.66667	2850	950

```
SQL>
```

HAVING

- ▶ where절과 having절, group by절을 모두 함께 사용할 경우
 1. where절에 명시된 조건을 만족하는 레코드들을 검색
 2. group by절에 명시된 필드의 값이 서로 일치하는 레코드들 끼리 그룹을 지어 집계 함수를 적용
 3. 마지막으로 그 집계 함수를 적용한 결과들 중에서 having절을 만족하는 결과만 골라서 출력

널(null)의 처리

- ▶ 널을 검색하는 방법
- ▶ 형식

<필드이름> **is null**
<필드이름> **is not null**

- ▶ 예) takes 테이블에서 아직 학점이 부여되지 않은 학생의 학번을 검색

(질의 64)

```
select  stu_id
from    takes
where   grade is null
```

널의 처리

- ▶ 예) takes 테이블에서 학점이 'A+'가 아닌 학생들의 학번을 검색

(질의 65)

```
select    stu_id
from      takes
where     grade <> 'A+'
```

- ▶ grade 필드의 값이 널인 레코드에 대해서는 질의 결과에 포함되지 않음
- ▶ 하지만 **count(*)**는 특정 필드가 아닌 레코드 전체에 대한 연산이므로 널의 존재 여부와는 무관함

중첩 질의(nested query)

- ▶ SQL문을 다른 SQL문 안에 중첩하여 사용하는 질의
- ▶ 복잡한 질의를 쉽게 표현할 수 있는 수단을 제공
- ▶ 내부질의(inner query), 부질의(subquery)
 - ▶ 내부에 포함된 SQL문
- ▶ 외부질의(outer query)
 - ▶ 부 질의를 내부적으로 갖는 SQL문
- ▶ 부 질의는 외부 질의의 from 절이나 where 절에 위치
- ▶ 종류
 - ▶ in, not in
 - ▶ =some, <=some, <some, >some, >=some, <>some (some 대신 any를 사용해도 됨)
 - ▶ =all, <=all, <all, >all, >=all, <>all
 - ▶ exists, not exists

IN, NOT IN

- ▶ 예) '301호' 강의실에서 개설된 강좌의 과목명을 출력

(질의 66)

```
select      title
from        course
where       course_id in
              (select distinct course_id
               from      class
               where     classroom = '301호')
```

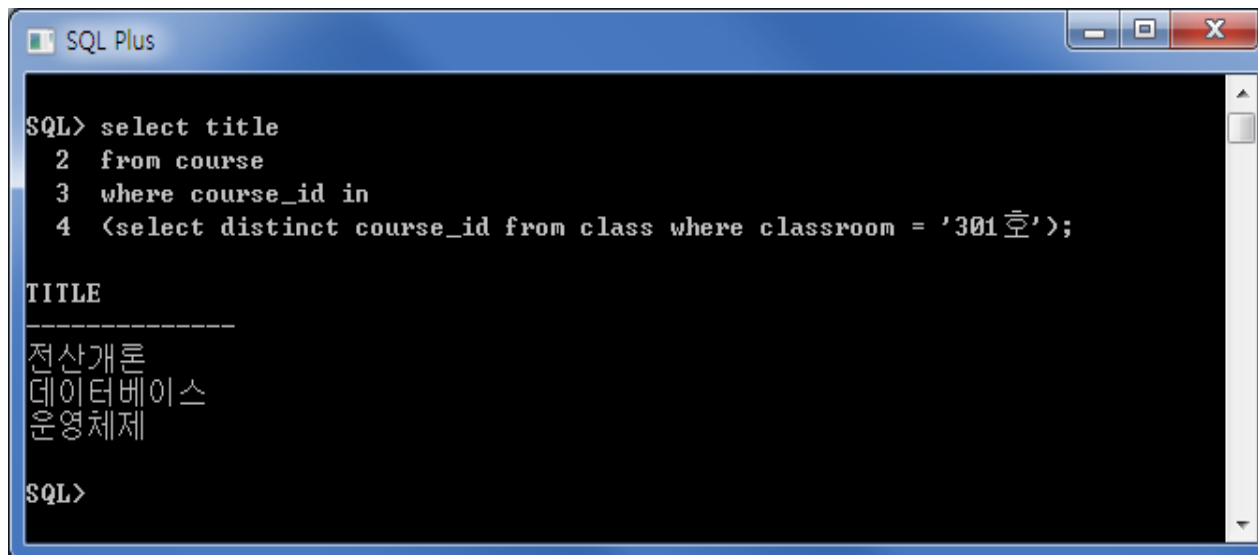
- ▶ 부 질의
 - ▶ 키워드 **in** 뒤에 나오는 SQL문으로서 class 테이블에서 강의실이 '301호'인 교과목 번호를 검색
- ▶ 외부 질의
 - ▶ course 테이블에서 course_id 필드의 값이 부 질의의 검색 결과에 포함되는 경우(**in**)에만 과목명을 출력

IN, NOT IN

▶ 동일한 표현

(질의 67)

```
select    distinct title
from      course c1, class c2
where      c1.course_id = c2.course_id and
            classroom = '301호'
```



The screenshot shows a window titled "SQL Plus" with a black background and white text. The text displays an SQL query and its results. The query is: `SQL> select title`, `2 from course`, `3 where course_id in`, `4 (select distinct course_id from class where classroom = '301호');`. Below the query, the results are shown under the heading "TITLE" with a dashed line separator. The results are: "전산개론", "데이터베이스", and "운영체제". The prompt "SQL>" is visible at the bottom.

```
SQL> select title
2  from course
3  where course_id in
4  (select distinct course_id from class where classroom = '301호');

TITLE
-----
전산개론
데이터베이스
운영체제

SQL>
```

IN, NOT IN

- ▶ 예) 2012년 2학기에 개설되지 않은 과목명을 검색

(질의 68)

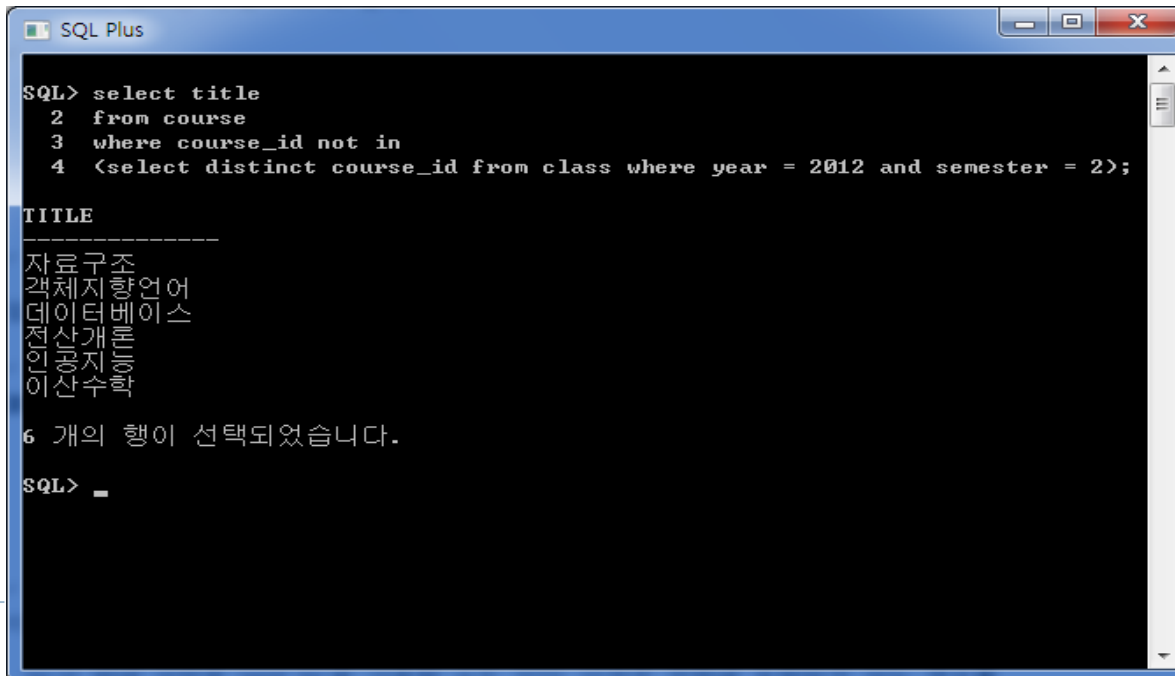
select
from
where

title
course
course_id

not in

(**select distinct** course_id
from class

where year = 2012 **and** semester = 2)



```
SQL> select title
2  from course
3  where course_id not in
4  <select distinct course_id from class where year = 2012 and semester = 2>;

TITLE
-----
자료구조
객체지향언어
데이터베이스
전산개론
인공지능
이산수학

6 개의 행이 선택되었습니다.

SQL> _
```

SOME, ALL

- ▶ =some

- ▶ 지정된 필드의 값이 부 질의 검색 결과에 존재하는 임의의 값과 같은지를 나타낼 때 사용
- ▶ in과 같은 의미

- ▶ <=some

- ▶ 부 질의의 검색 결과에 존재하는 임의의 값보다 작거나 같은지를 나타낼 때 사용

- ▶ =all

- ▶ 지정된 필드의 값이 부 질의 검색 결과에 포함된 모든 값과 같은지를 판단

- ▶ <=all

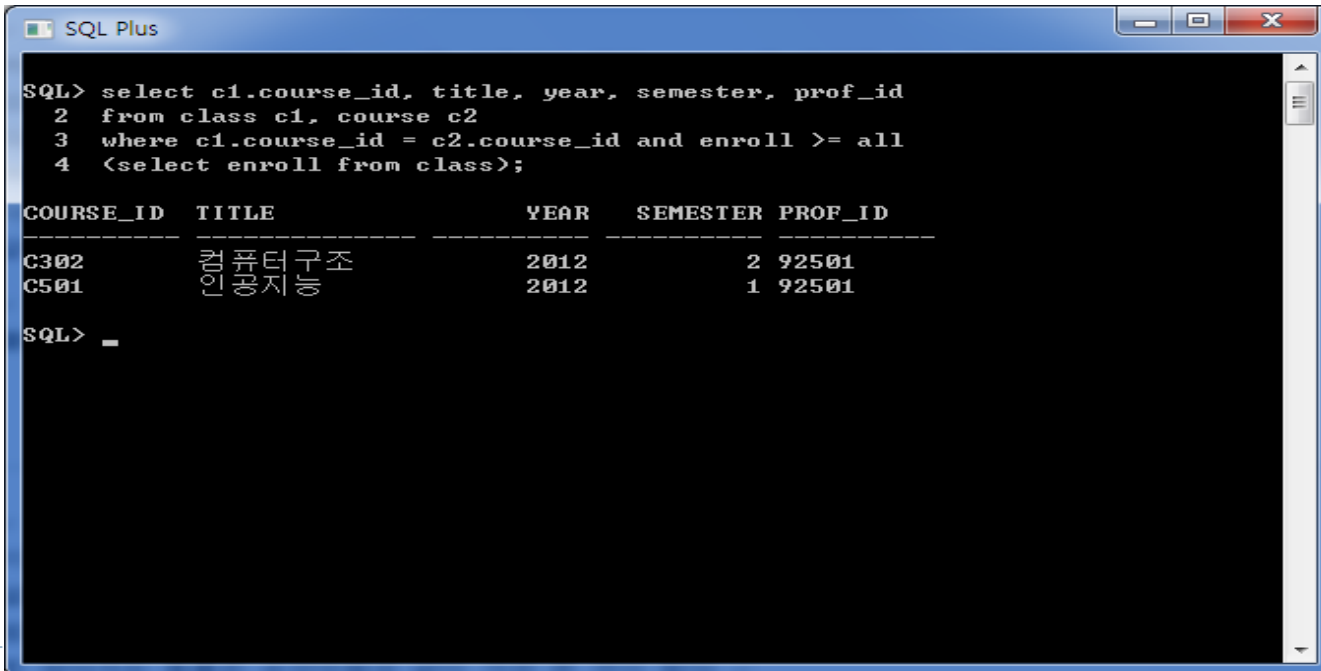
- ▶ 지정된 필드의 값이 부 질의 검색 결과에 포함된 모든 값보다 작거나 같은지를 판단

SOME, ALL

- ▶ 예) 가장 많은 수강 인원을 가진 강좌를 검색

(질의 69)

```
select    c1.course_id, title, year, semester, prof_id
from      class c1, course c2
where     c1.course_id = c2.course_id and enroll >= all
                                                (select enroll from class)
```



```
SQL> select c1.course_id, title, year, semester, prof_id
2  from class c1, course c2
3  where c1.course_id = c2.course_id and enroll >= all
4  <select enroll from class>;
```

COURSE_ID	TITLE	YEAR	SEMESTER	PROF_ID
C302	컴퓨터구조	2012	2	92501
C501	인공지능	2012	1	92501

```
SQL> _
```

EXISTS, NOT EXIST

- ▶ 부 질의 검색 결과에 최소한 하나 이상의 레코드가 존재하는지의 여부를 표현
- ▶ exists
 - ▶ 최소한 한 개의 레코드가 존재하면 참이 되고 그렇지 않으면 거짓
- ▶ not exists
 - ▶ 부 질의의 결과에 레코드가 하나도 없으면 참이 되고 하나라도 존재하면 거짓
- ▶ 예) '301호' 강의실에서 개설된 강좌의 과목명을 출력

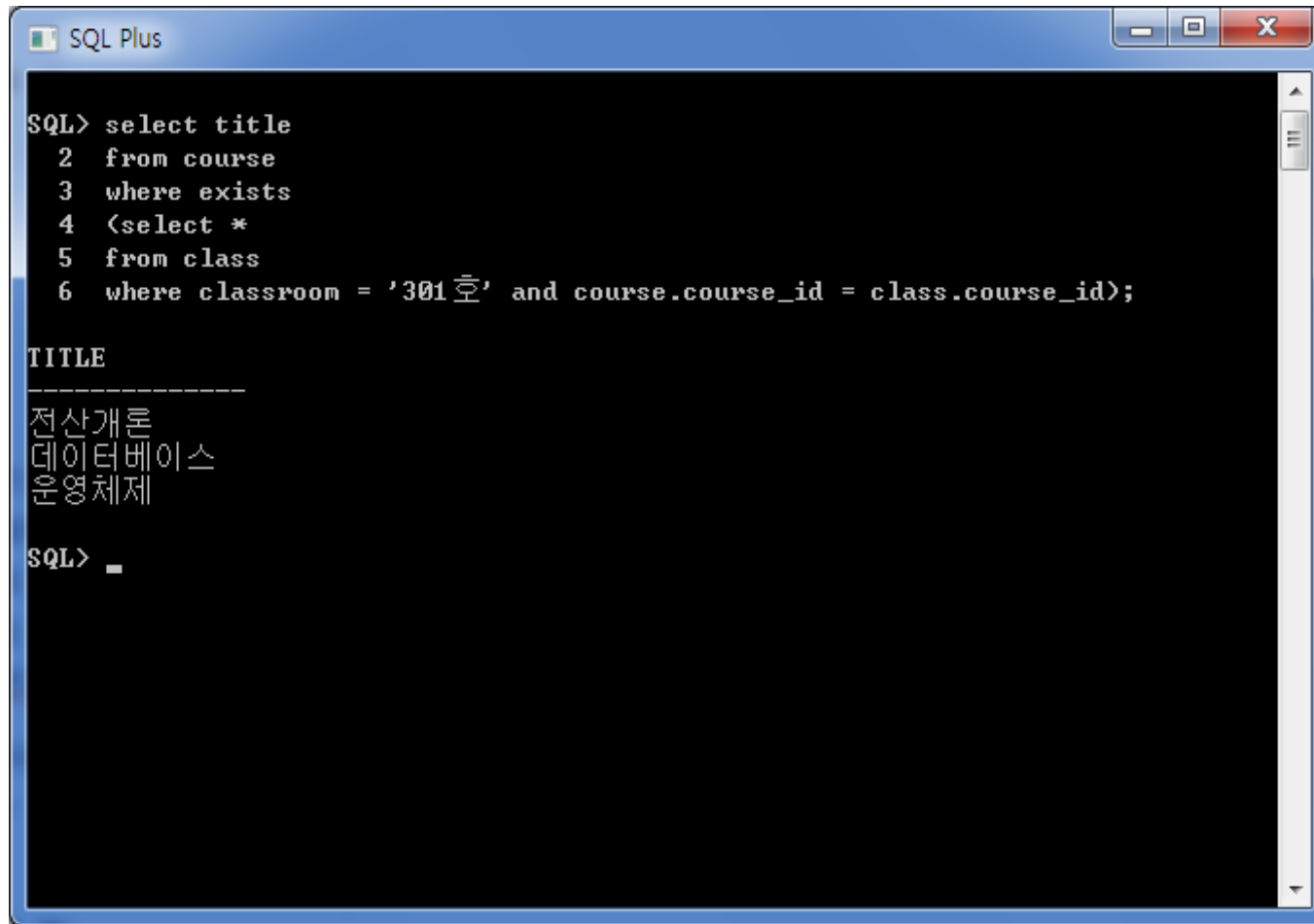
(질의 70)

select
from
where

title
course
exists
(**select** *
from class
where

classroom = '301호' **and**
course.course_id = class.course_id)

EXISTS, NOT EXISTS



```
SQL> select title
  2   from course
  3  where exists
  4   (<select *
  5    from class
  6   where classroom = '301호' and course.course_id = class.course_id);

TITLE
-----
전산개론
데이터베이스
운영체제

SQL> _
```

EXISTS, NOT EXIST

- ▶ (질의 68)을 not exists로 표현 가능

(질의 71)

```
select title
from   course
where  not exists
       (select *
        from   class
        where  year = 2012 and
               semester = 2 and
               course.course_id = class.course_id)
```

뷰 (view)

- ▶ 기존 테이블들로부터 생성되는 **가상의 테이블**
- ▶ 테이블처럼 물리적으로 생성되는 것이 아니라 기존의 테이블들을 조합하여 사용자에게 실제로 존재하는 테이블인 것처럼 보이게 함
- ▶ 기능
 - ▶ 특정 사용자에게 테이블의 내용 중 일부를 숨길 수 있기 때문에 보안의 효과
 - ▶ 복잡한 질의의 결과를 뷰로 만들어서 사용하게 되면 질의를 간단히 표현 할 수 있음

뷰 생성

- ▶ 생성된 뷰는 테이블과 동등하게 사용
- ▶ 형식

create or replace view	<뷰이름> <select문>	as
-------------------------------	--------------------	-----------

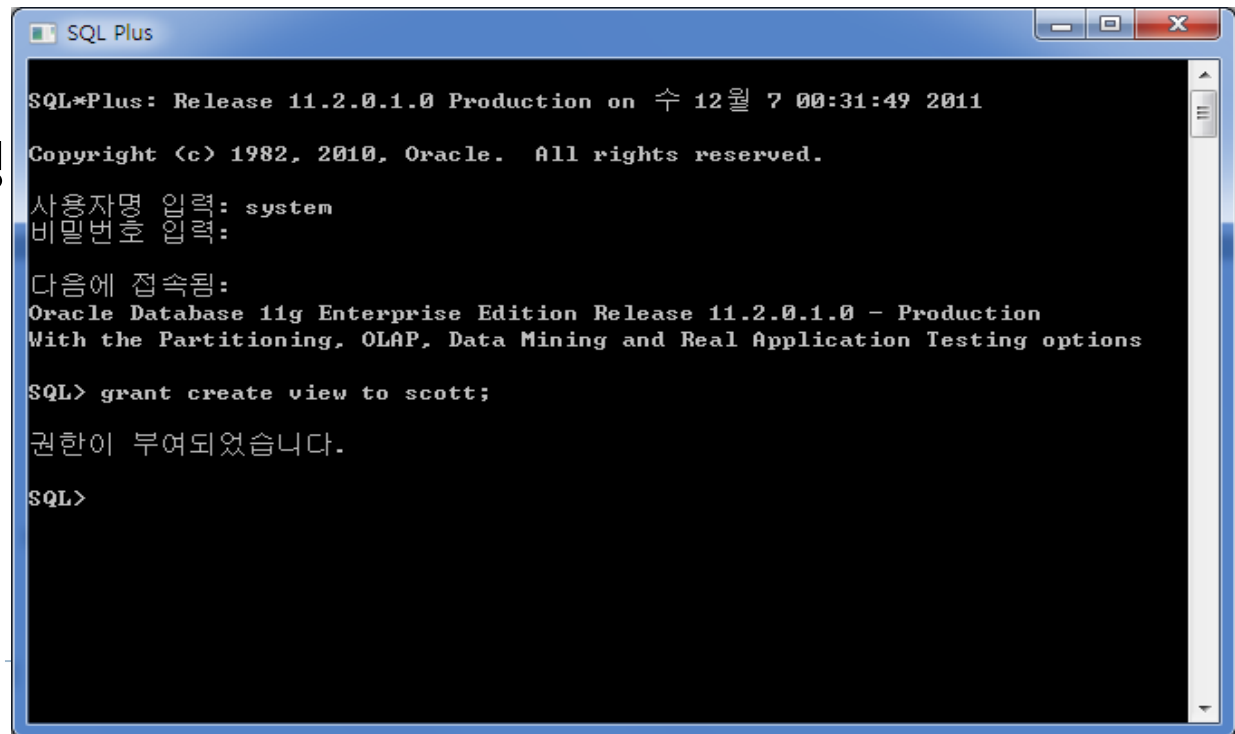
- ▶ **or replace** 키워드를 추가하면 <뷰이름>과 같은 뷰가 이미 존재하는 경우 기존의 뷰를 지우고 새로 생성
- ▶ <select문>
 - ▶ 뷰 생성에 사용될 **select**문

뷰 생성

- ▶ 대부분의 DBMS에서는 사용자 계정에는 뷰 생성 권한이 부여되지 않음
- ▶ 관리자 계정이 아닌 사용자 계정으로 로그인하여 뷰를 생성하려면 뷰 생성과 관련된 권한이 부여되어야 함
- ▶ 오라클에서 뷰 생성 권한을 부여하기 위한 형식

grant create view to <사용자 계정>

scott에게 뷰 생성
권한의 부여



```
SQL*Plus: Release 11.2.0.1.0 Production on 수 12월 7 00:31:49 2011
Copyright (c) 1982, 2010, Oracle. All rights reserved.

사용자명 입력: system
비밀번호 입력:

다음에 접속됨:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> grant create view to scott;

권한이 부여되었습니다.

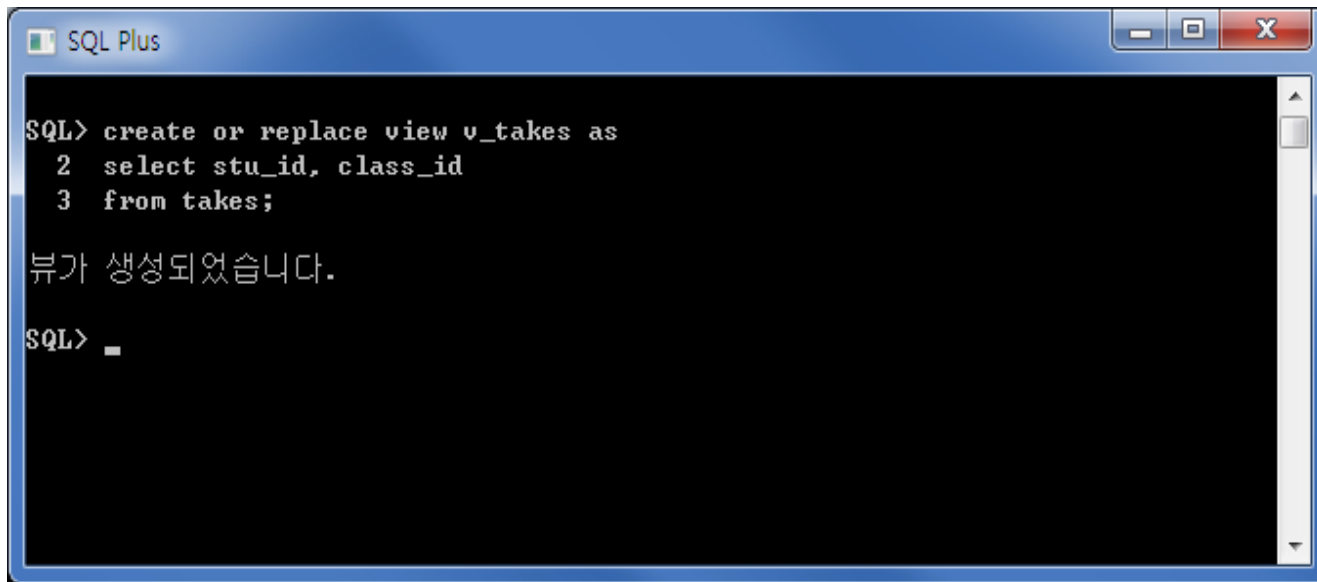
SQL>
```

뷰 생성

- ▶ 예) takes 테이블에서 grade 필드를 제외한 나머지 필드만으로 구성된 뷰를 생성

(질의 72)

```
create or replace view v_takes as
select stu_id, class_id
from takes
```



The screenshot shows a SQL Plus window with a blue title bar and standard Windows window controls. The main area is a black terminal with white text. It displays the SQL command to create or replace the view v_takes, followed by the confirmation message '뷰가 생성되었습니다.' and the next prompt.

```
SQL Plus

SQL> create or replace view v_takes as
  2  select stu_id, class_id
  3  from takes;

뷰가 생성되었습니다.

SQL> _
```

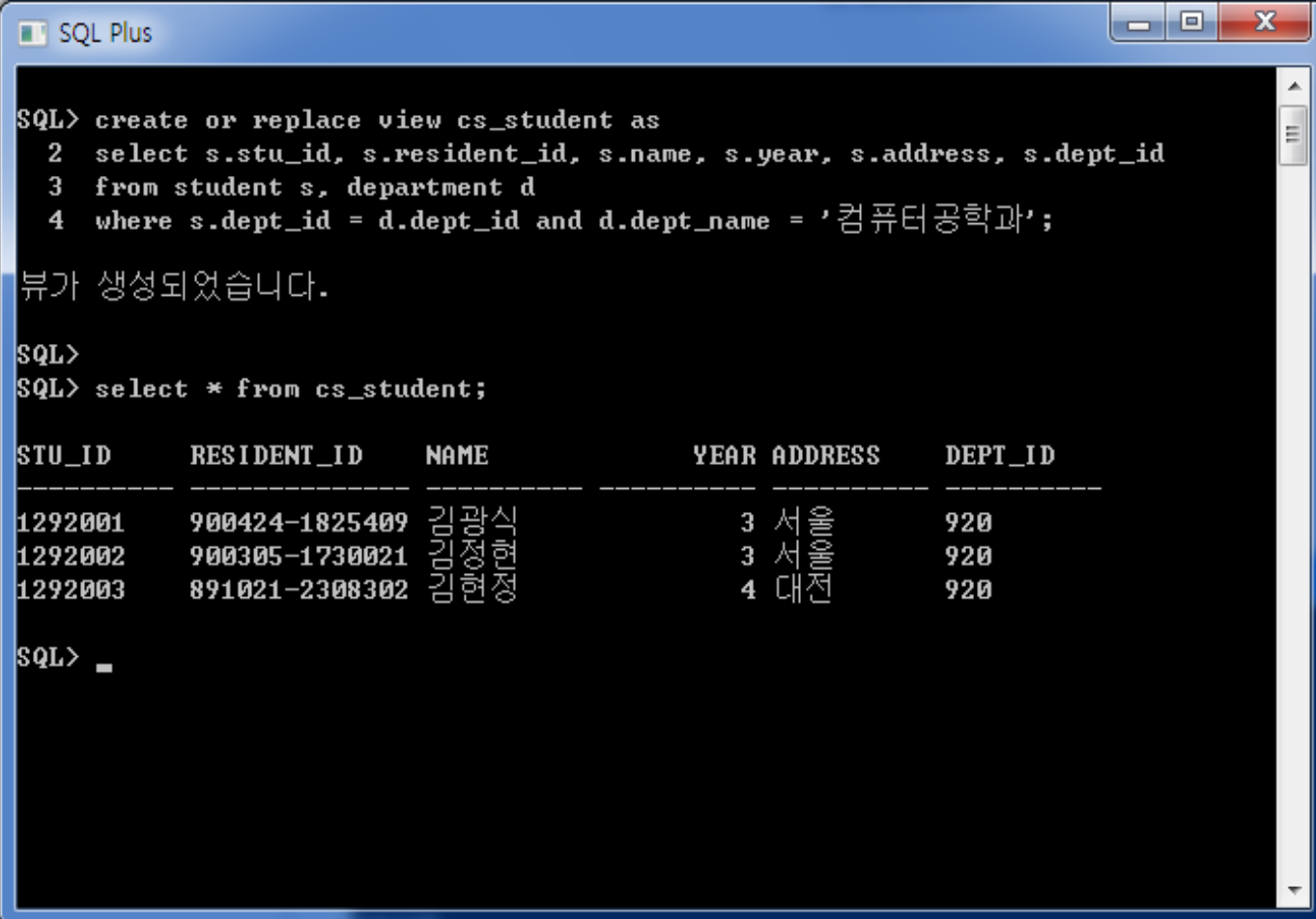
뷰 생성

- ▶ 예) student 테이블에서 컴퓨터공학과 학생들 레코드만 추출하여 뷰를 생성

(질의 73)

```
create or replace view cs_student as  
  select    s.stu_id, s.resident_id, s.name, s.year, s.address, s.dept_id  
  from      student s, department d  
  where     s.dept_id = d.dept_id and  
            d.dept_name = '컴퓨터공학과'
```

뷰 생성



```
SQL> create or replace view cs_student as
  2  select s.stu_id, s.resident_id, s.name, s.year, s.address, s.dept_id
  3  from student s, department d
  4  where s.dept_id = d.dept_id and d.dept_name = '컴퓨터공학과';

뷰가 생성되었습니다.

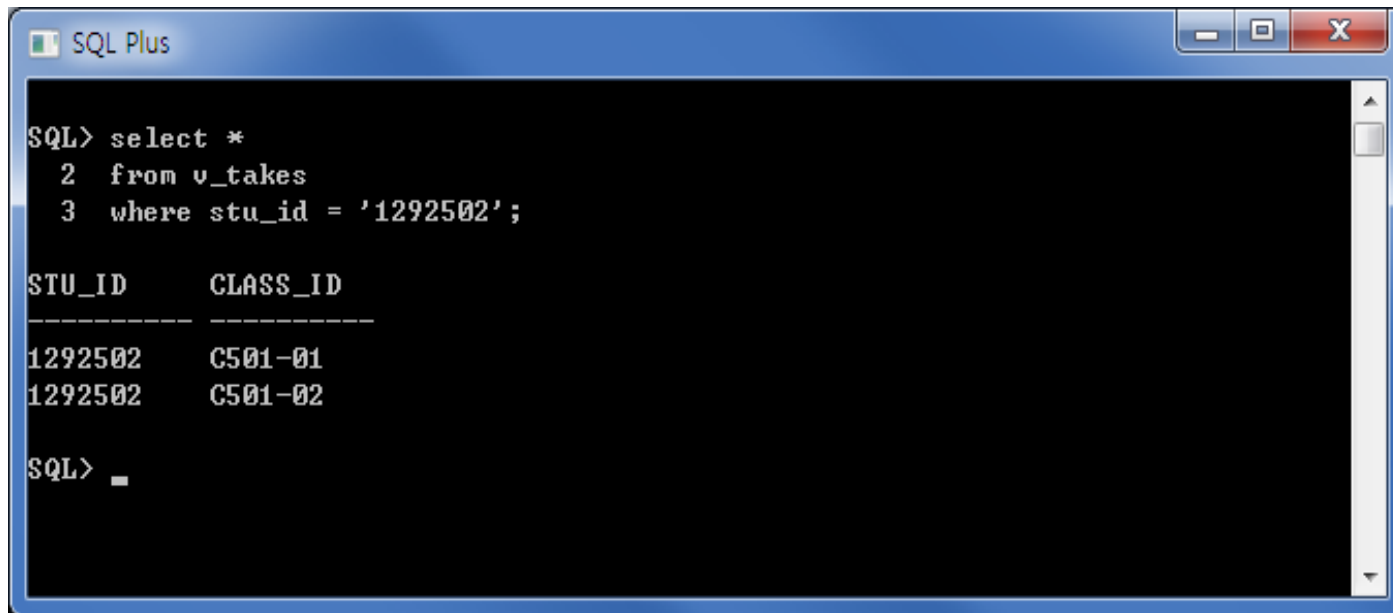
SQL>
SQL> select * from cs_student;
```

STU_ID	RESIDENT_ID	NAME	YEAR	ADDRESS	DEPT_ID
1292001	900424-1825409	김광식	3	서울	920
1292002	900305-1730021	김정현	3	서울	920
1292003	891021-2308302	김현정	4	대전	920

```
SQL> _
```

뷰 사용

- ▶ 예) v_takes 뷰에 대해 select문을 실행



The screenshot shows a SQL Plus window with a blue title bar and standard Windows window controls. The main area is black with white text. It displays an SQL query and its output. The query is: `SQL> select *
2 from v_takes
3 where stu_id = '1292502';`. The output is a table with two columns: `STU_ID` and `CLASS_ID`. The first row of data shows `1292502` and `C501-01`, and the second row shows `1292502` and `C501-02`. The prompt `SQL>` is followed by a cursor line.

```
SQL> select *  
2 from v_takes  
3 where stu_id = '1292502';  
  
STU_ID      CLASS_ID  
-----  
1292502     C501-01  
1292502     C501-02  
  
SQL> _
```

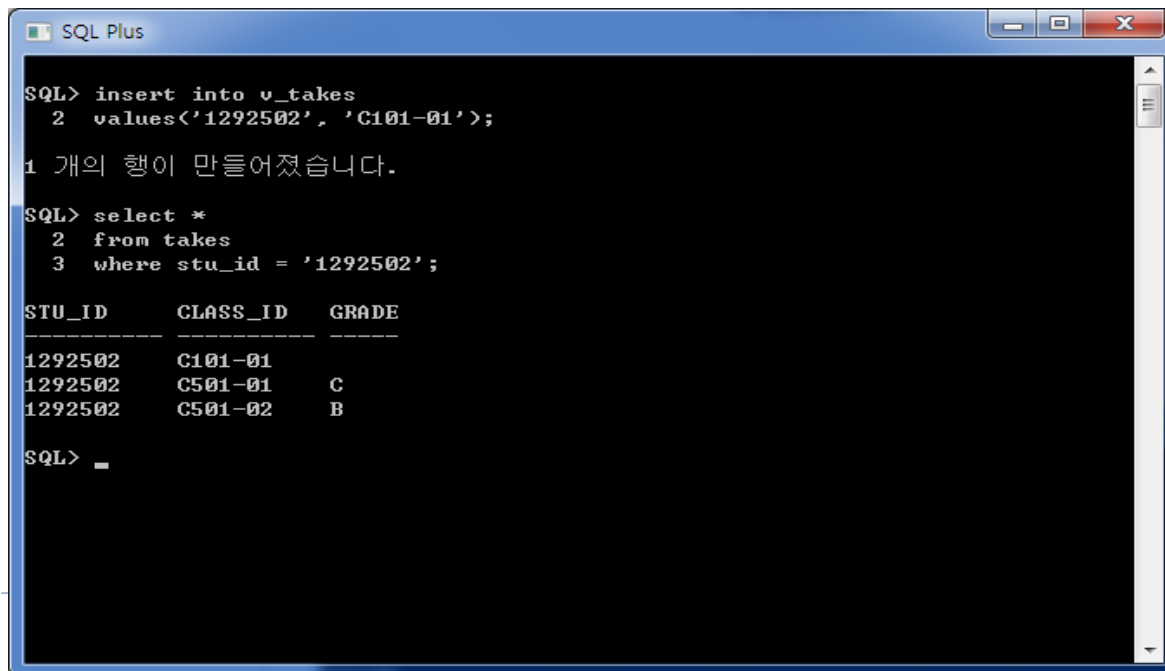
뷰 사용

- ▶ 뷰에 대해서 **insert, update, delete**문을 실행
- ▶ 예) v_takes 뷰에 대해 레코드를 삽입

(질의 74)

```
insert into      v_takes
values ('1292502', 'C101-01')
```

- ▶ v_takes 뷰에 포함되지 않은 grade 필드에는 널이 삽입



The screenshot shows a SQL Plus window with the following content:

```
SQL> insert into v_takes
2  values(<'1292502', 'C101-01');

1 개의 행이 만들어졌습니다.

SQL> select *
2  from takes
3  where stu_id = '1292502';
```

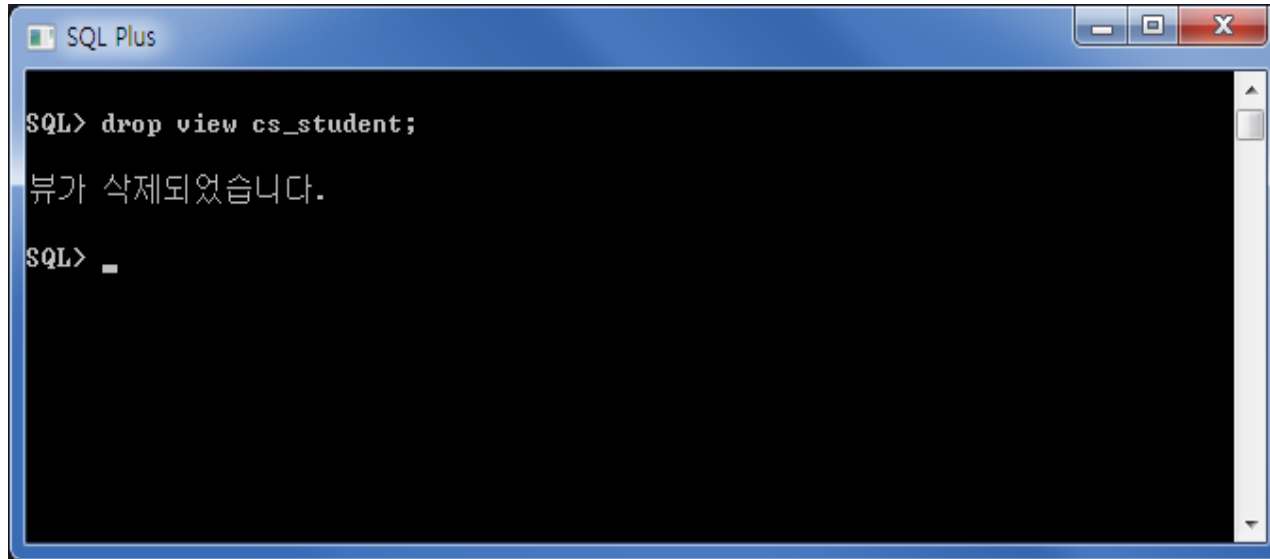
STU_ID	CLASS_ID	GRADE
1292502	C101-01	
1292502	C501-01	C
1292502	C501-02	B

SQL> _

뷰 사용

▶ 형식

drop view <뷰이름>



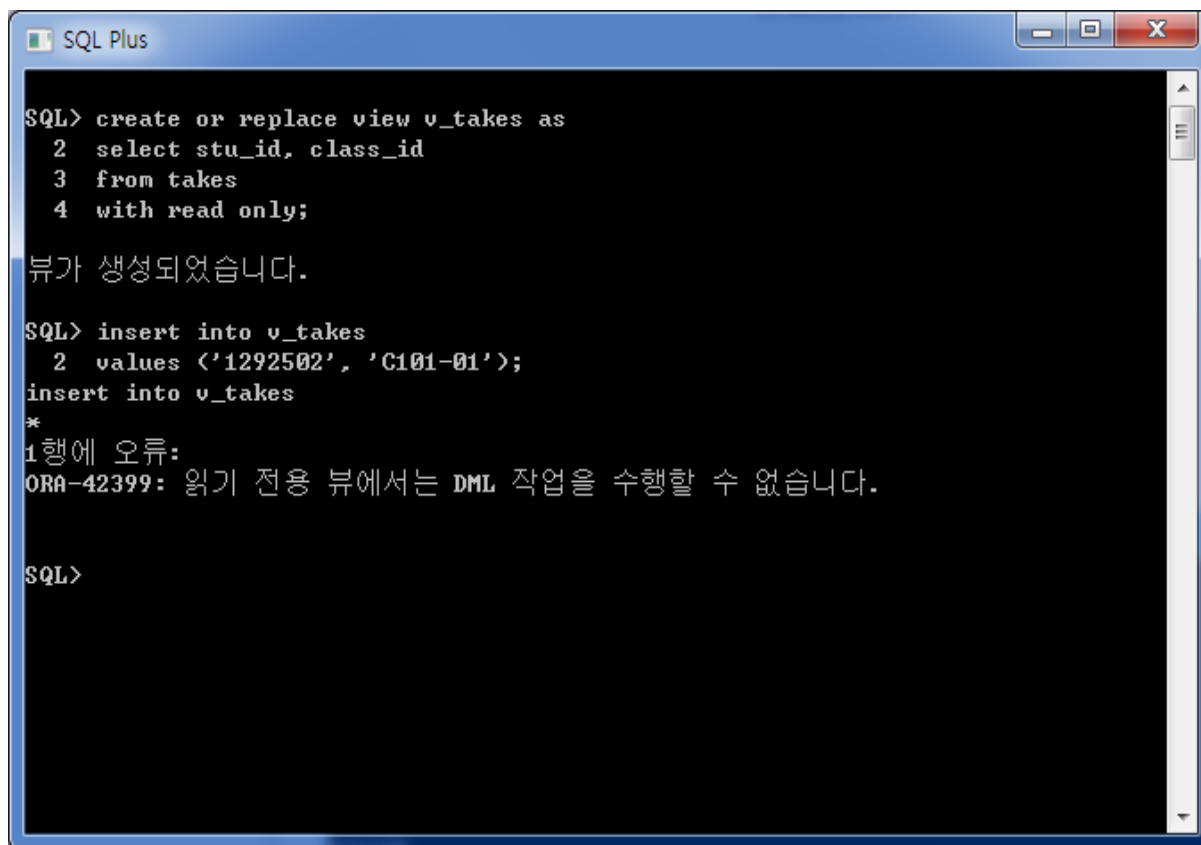
The screenshot shows a window titled "SQL Plus" with a black background and white text. The text inside the window is as follows:

```
SQL> drop view cs_student;  
뷰가 삭제되었습니다.  
SQL> _
```


뷰 삭제

▶ 읽기 전용 뷰

- ▶ 뷰를 생성할 때 insert, update, delete문과 같은 데이터 조작 언어의 사용을 불가능하게 하려면, with read only 키워드를 추가



```
SQL> create or replace view v_takes as
  2  select stu_id, class_id
  3  from takes
  4  with read only;

뷰가 생성되었습니다.

SQL> insert into v_takes
  2  values ('1292502', 'C101-01');
insert into v_takes
*
1 행에 오류:
ORA-42399: 읽기 전용 뷰에서는 DML 작업을 수행할 수 없습니다.

SQL>
```