

3장. SQL 소개

이 동 호

데이터베이스 연구실
소프트웨어학부

- 3.1 SQL 질의 언어의 개요
- 3.2 SQL 데이터 정의
- 3.3 SQL 질의의 기본 구조
- 3.4 추가적인 기본 연산
- 3.5 집합 연산
- 3.6 널 값
- 3.7 집계 함수
- 3.8 중첩 하위 질의
- 3.9 데이터베이스의 변경

3.1 SQL 질의 언어의 개요

- SQL 질의 언어는 다음과 같은 부분들로 구성
 - 데이터 정의 언어(**DDL**): 스키마 구조를 정의하는 명령어로 구성
 - 데이터 조작 언어(**DML**): 데이터베이스에 질의를 하는 명령어로 구성
 - 무결성 **Integrity**
 - SQL DDL은 데이터가 지켜야 할 무결성을 명세하는 명령어를 포함
 - 뷰(**View**) 정의
 - SQL DDL 은 뷰를 정의하는 명령어를 포함
 - 트랜잭션 제어(**Transaction control**)
 - 내장 **SQL(Embedded SQL)**
 - 인증(**Authorization**)

3.2 SQL 데이터 정의

SQL 데이터 정의 언어(DDL)는 아래와 같은 릴레이션에 대한 정보를 명세할 수 있다.

- 각 릴레이션의 스키마
- 각 속성과 연계된 값들의 도메인
- 무결성 제약조건
- 이외,
 - 각 릴레이션을 위한 색인 정보
 - 각 릴레이션을 위한 보안과 권한 정보
 - 각 릴레이션들의 물리적 저장 구조

SQL DDL에 사용되는 기본 타입

- **char(n)** - n개의 고정된 길이를 갖는 문자열
- **varchar(n)** - 최대 n개의 길이를 갖는 가변 길이 문자열
- **int** - 정수 (기계 종속적인 정수의 유한집합).
- **smallint** - 작은 정수(정수 도메인 타입의 기계 종속적인 부분집합)
- **numeric(p,d)** - 정확도 p(p개의 숫자, 소수점 이하 d자리) 갖는 고정 소수점 수
 - numeric (3,1) : 44.5 → OK, 444.5 or 0.32 → ?
- **real, double precision** - 기계 종속적인 부동 소수점 수와 두배 정확도를 갖는 부동 소수점 수
- **float(n)** - 적어도 n개 숫자의 정확도를 갖는 부동 소수점 수
- 4장에서 더 자세히 다룰 예정

기본 스키마 정의 – 테이블 생성

- 릴레이션은 **create table** 명령어로 생성

```
create table r (A1 D1, A2 D2, ..., An Dn,  
              (무결성-제약조건1),  
              ...,  
              (무결성-제약조건k))
```

- *r* : 릴레이션 이름
- *A*_{*i*} : 릴레이션 *r*에 있는 속성 이름
- *D*_{*i*} : 속성 *A*_{*i*}의 도메인에 존재하는 값의 데이터 타입

- 예제:

```
create table instructor (  
    ID          char(5),  
    name       varchar(20) not null,  
    dept_name varchar(20),  
    salary    numeric(8,2))
```

- insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- insert into** *instructor* **values** ('10211', null, 'Biology', 66000);

Create Table에서 무결성-제약조건 명시

```
create table instructor (  
    ID                char(5),  
    name              varchar(20) not null,  
    dept_name varchar(20),  
    salary           numeric(8,2),  
    primary key (ID),  
    foreign key (dept_name) references department)
```

- **not null**
- **primary key** ($A_1, \dots, A_n$)
 - 주키는 자동으로 널이 아님을 선언하는 효과도 있음
- **foreign key** ($A_m, \dots, A_n$) **references** r
 - 속성 ($A_m, \dots, A_n$)는 릴레이션 r 의 주키로 나와야 한다.
 - 위 예에서 *dept_name*는 *department*의 주키이어야 한다는 것을 선언한 것임

릴레이션 정의 (추가 예제)

- **create table** *student* (
 ID **varchar**(5),
 name **varchar**(20) **not null**,
 dept_name **varchar**(20),
 tot_cred **numeric**(3,0),
 primary key (*ID*),
 foreign key (*dept_name*) **references** *department*));
- **create table** *takes* (
 ID **varchar**(5),
 course_id **varchar**(8),
 sec_id **varchar**(8),
 semester **varchar**(6),
 year **numeric**(4,0),
 grade **varchar**(2),
 primary key (*ID*, *course_id*, *sec_id*, *semester*, *year*),
 foreign key (*ID*) **references** *student*,
 foreign key (*course_id*, *sec_id*, *semester*, *year*) **references** *section*));
- 참고: *sec_id* 는 주키에서 빠질 수도 있다. → 학생은 동일한 학기, 동일한 과목의 분반에 동시에 등록할 수 없기 때문임

추가 예제

- **create table** *course* (
 course_id **varchar(8) primary key**,
 title **varchar(50)**,
 dept_name **varchar(20)**,
 credits **numeric(2,0)**,
 foreign key (*dept_name*) **references** *department*));
- 주키 선언은 무결성-제약조건 형태가 아니라 속성 선언시 함께 명시될 수 있음

테이블 삭제와 변경

- **drop table** *student*
 - *student* 테이블을 삭제하고 내용도 삭제
- **delete from** *student*
 - *student* 테이블의 내용은 삭제하지만 테이블은 유지
- **alter table**
 - **alter table** *r* **add** *A D*
 - *A* : 릴레이션 *r*에 추가할 속성의 이름
 - *D* : *A*의 도메인
 - 모든 튜플은 새로운 속성의 초기값으로 널 값이 세팅됨
 - **alter table** *r* **drop** *A*
 - 릴레이션 *r*로부터 *A*속성을 삭제
 - 속성 삭제는 많은 데이터베이스에서 제공되지 않음

3.3 SQL 질의의 기본 구조

- SQL (**DML**) 은 질의 정보를 제공하거나 튜플을 삽입, 삭제, 갱신할 있으
- 전형적인 SQL 질의형태:

```
select  $A_1, A_2, \dots, A_n$ 
from  $r_1, r_2, \dots, r_m$ 
where  $P$ 
```

- A_i : 속성
 - R_i : 릴레이션
 - P : 술어(predicate) – 질의 조건
- SQL 질의의 결과 → 릴레이션

select절

- **select** 절은 질의 결과로 포함되는 속성들을 표시한다
 - 관계 대수의 추출 연산에 대응됨
- 예제 – find the names of all *instructors*:

```
select name  
from instructor
```
- 주의: SQL에서 이름은 대소문자를 구분하지 않음
 - 즉, *Name* $\equiv$ *NAME* $\equiv$ *name*

select절 (Cont.)

- SQL 은 질의 결과뿐만 아니라 릴레이션에 중복을 허용함
 - 중복을 제거하기 위해서는 **select** 다음에 **distinct** 를 명시해야 함

- Find the names of all departments with instructor (중복제거)

```
select distinct dept_name  
from instructor
```

- 중복을 제거하고 싶지 않을 경우에는 **select** 다음에 **all**을 명시

```
select all dept_name  
from instructor
```

distinct 나 **all**을 쓰지 않으면? 즉, default는 ?

→ Oracle에서는 중복된 것이 있으면 모두 나온다. 즉, **all**이 default로 되어 있음

select절 (Cont.)

- 모든 속성을 표시할 때는 * (asterisk)를 사용

```
select *  
from instructor
```

- select** 절은 속성이나 상수 등에 산술연산자(+, -, *, and /)를 적용할 수 있음
- 예제 :

```
select ID, name, salary/12  
from instructor
```

instructor 테이블에서 연봉이 12로 나누어진 결과를 볼 수 있음

- The **from**절은 질의와 관련된 릴레이션을 명시함
 - 관계대수에서는 카티션 곱 연산에 해당
- Find the Cartesian product *instructor X teaches*
 - select ***
 - from *instructor, teaches***
 - 모든 가능한 *instructor – teaches*쌍을 만들어 냄
 - 너무 많은 결과를 생성해 냄
- 카티션 곱 자체는 **where**절(관계대수에서 선택 연산)과 연결되었을 때 유용하게 사용될 수 있음

- **where**절은 질의결과가 만족해야할 조건을 명시함
 - 관계대수에서 선택연산에 상응함
- To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```
- 개별 조건들은 논리연산자(**and**, **or**, and **not**)들로 연결 가능

카디션 곱: *instructor X teaches*

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

조인

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```
select section.course_id, semester, year, title
from section, course
where section.course_id = course.course_id and
dept_name = 'Comp. Sci.'
```

dept_name 앞에
Relation 이름이 없네?



자연조인

- 자연 조인은 공통된 속성들에 대해서 그 값이 서로 같은 튜플들만으로 가
지고 온다(공통된 속성중에서 하나만 유지함)
- select ***
from instructor natural join teaches;

1. 자연조인시 하나의 **table**이 생성된다
2. 조인에 사용된 속성 중 하나는 사라진다(중복되기 때문)
3. 자연조인 결과에 있는 속성을 가리키기 위해 원래의
릴레이션 이름을 포함한 속성이름 사용은 불가! (*instructor.name*)

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

자연조인(Cont.)

- List the names of instructors along with the course ID of the courses that they taught.
 - **select** *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID;*
 - **select** *name, course_id*
from *instructor natural join teaches;*

두 질의의 결과는 같은데, 그럼 뭐가 다른 거지?

일반적인 SQL 질의 처리 단계

1. **from** 절을 먼저 처리하여 새로운 **relation**을 만들어 냄
2. 1의 과정에서 생성된 새로운 **relation**에 **where**절을 적용함
3. 2의 결과로 얻은 각 **tuple**들에 대하여 **select** 처리를 함.

자연조인(Cont.)

- 자연조인의 위험!! – 이름은 동일한데 서로 관련이 없는 속성들이 잘 못 조인될 수 있음
- List the names of instructors along with the titles of courses that they teach
 - 잘못된 버전!!
 - **select name, title**
from instructor natural join teaches natural join course;

1. instructor와 teaches의 자연조인 : 공통속성 - ID
(ID, name, dept_name, salary, course_id, sec_id, ...)
2. 1의 결과와 course의 자연조인 : 공통속성 – course_id, dept_name
 - course_id뿐만 아니라 dept_name도 같아야 한다는 것을 요구함
 - 결국, 2의 결과는 “교수가 소속된 학과에서 가르치는 과목”만 가져옴
 - 교수가 다른 학과에서 가르치는 과목은 ?
 - Instructor에 있는 dept_name과 course에 있는 dept_name은 같은 속성이름이지만 의미적으로 다르게 사용되기 때문임

자연조인 (Cont.)

– 올바른 버전1

- **select** *name, title*
from *instructor* **natural join** *teaches, course*
where *teaches*.*course_id* = *course.course_id*;

→ *course_id*가 자연조인 결과에도 있고, *course*에도 있기 때문에 ambiguity를 피하기 위해 relation이름을 붙임

– 올바른 버전2

- **select** *name, title*
from (*instructor* **natural join** *teaches*)
join *course* **using** (*course_id*);

- 질의는 교수이름과 교수가 강의하는 모든 과목의 과목명을 가져옴
- 자연조인은 기본적으로 두 릴레이션에 존재하는 모든 공통속성을 이용하여 조인하기 때문에 조인 속성을 지정하기 위하여 **join ~ using (조인속성)**을 이용

3.4 추가 기본 질의 – Rename 연산

- SQL은 **as**절을 이용하여 릴레이션이나 속성의 이름을 재명명할 수 있음
old-name as new-name
- 예제,
 - select** *ID, name, salary/12 as monthly_salary*
from *instructor*
- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
 - select distinct** *T. name*
from *instructor as T, instructor as S*
where *T.salary > S.salary and S.dept_name = 'Comp. Sci.'*

T				S			
ID	name	dept_name	salary	ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000	10101	Srinivasan	Comp. Sci.	<u>65000</u>
12121	Wu	Finance	90000	12121	Wu	Finance	<u>90000</u>
15151	Mozart	Music	40000	15151	Mozart	<u>Music</u>	<u>40000</u>
22222	Einstein	Physics	95000	22222	Einstein	Physics	<u>95000</u>
32343	El Said	History	60000	32343	El Said	History	<u>60000</u>

3.4 추가 기본 질의 – Rename 연산

- **as** 는 생략될 수 있음
$$\text{instructor as } T \equiv \text{instructor } T$$
 - 오라클에서는 **as**는 반드시 생략됨

문자열 연산

- SQL 은 문자열 비교를 위하여 문자열 매칭 연산자를 제공함. “like” 연산자는 두개의 특별문자를 사용하여 문자열 패턴 매칭을 사용할 수 있음
 - 퍼센트(%) : 부분문자열과 매칭됨
 - 밑줄(_). 하나의 문자에 매칭됨

- Find the names of all instructors whose name includes the substring “dar”.

```
select name
from instructor
where name like '%dar%'
```

- “100 %” 문자열 매칭을 위해서 탈출문자(\) 사용


```
like '100 \%' escape '\'
```

- %를 special 문자가 아니라 일반 문자로 처리
- 그래서, 100 %으로 시작되는 모든 문자열
- like 'ab\%cd%' escape '\' → “ab%cd”로 시작되는 모든 문자열

문자열 연산(Cont.)

- 문자열 패턴은 대소문자 구분함
- 문자열 패턴 매칭의 예:
 - ‘Intro%’ : “Intro”로 시작하는 모든 문자열
 - ‘%Comp%’ : “Comp”를 포함하는 모든 문자열
 - ‘___’ : 정확히 3개의 문자들로 구성된 문자열과 매칭
 - ‘___ %’ : 적어도 3개의 문자들로 구성된 문자열과 매칭
- SQL은 다양한 문자열 연산을 제공함
 - 문자열 합침 (using “||”)
 - 대소문자 변환
 - 문자열 길이, 부분문자열 추출 등등

튜플 전개 순서화 (**order by**)

- List in alphabetic order the names of all instructors

```
select distinct name  
from   instructor  
order by name
```

- 내림차순은 **desc** , 오름차순은 **asc** 사용. 디폴트는 오름차순
 - 예제: **order by** *name desc*
- 다수의 속성에 대해서 사용 가능
 - 예제: **order by** *dept_name, name*

- **between** 비교 연산자

- 예제: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq \$90,000$ and $\leq \$100,000$)
- **select** *name*
from *instructor*
where *salary* **between** 90000 **and** 100000

- where salary $\geq$ 90000 and salary $\leq$ 100000

- 튜플 비교

- **select** *name, course_id*
from *instructor, teaches*
where (*instructor.ID, dept_name*) = (*teaches.ID*, 'Biology');

- where instructor.ID = teaches.ID and dept_name = 'Biology'

참고: not between 연산자도 있음

3.5 집합 연산

- Find courses that ran in Fall 2009 or in Spring 2010
 (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
union
 (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)
- Find courses that ran in Fall 2009 and in Spring 2010
 (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
intersect
 (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)
- Find courses that ran in Fall 2009 but not in Spring 2010
 (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
except
 (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

집합 연산 (Cont.)

- **union, intersect, and except**
 - 상기 연산자는 자동으로 중복을 제거함
- **union all, intersect all and except all.**
 - 중복을 허용

3.6 널(Null) 값

- 튜플의 속성들이 널 값을 갖는 것을 허용
- 널 값은 “알 수 없는 값(unknown)” 혹은 “존재하지 않는 값(not exist)”을 의미함
- 널 값을 포함한 산술 연산의 결과는 널이다.
 - Example: $5 + \text{null} \rightarrow \text{null}$
- “is null”은 널 값을 검사할 때 사용됨
 - 예 제: Find all instructors whose salary is null.

```
select name
from instructor
where salary is null
```

잠깐, “salary = null” 과연 유효한 표현인가 ?
SQL에서 사용가능한가?

null은 값을 모르거나 존재하지 않을 경우를 의미함. So, salary = null은 salary 값이 null이라는 것을 알고 있음을 의미. 의미적으로 = 은 맞지 않음. 그러나 ‘=’ 앞에 있는 expression이 null로 평가되면 true를 return하기도 함 참고로 Oracle은 허용함

널 값(Cont.)

- 널 값을 포함한 비교 연산은 *unknown*으로 처리함
 - Example: $5 < \text{null}$ or $\text{null} <> \text{null}$ or $\text{null} = \text{null}$
- *unknown*으로 처리되는 논리 예제:
 - OR: $(\text{unknown} \text{ or } \text{true}) = \text{true}$,
 $(\text{unknown} \text{ or } \text{false}) = \text{unknown}$
 $(\text{unknown} \text{ or } \text{unknown}) = \text{unknown}$
 - AND: $(\text{true} \text{ and } \text{unknown}) = \text{unknown}$,
 $(\text{false} \text{ and } \text{unknown}) = \text{false}$,
 $(\text{unknown} \text{ and } \text{unknown}) = \text{unknown}$
 - NOT: $(\text{not unknown}) = \text{unknown}$
 - 만약 P 가 *unknown*이면 “ P is **unknown**”은 *true*로 처리
- **where**절의 조건이 만약 *unknown*으로 평가되면 *false*로 처리됨

3.7 집계 함수

- 집계 함수는 릴레이션에서 열(즉, 속성) 값들에 대해서 동작을 하면 특정 값을 반환한다.

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

집계 함수(Cont.)

- Find the average salary of instructors in the Computer Science department
 - **select avg** (*salary*)
from *instructor*
where *dept_name*= 'Comp. Sci.';
- Find the total number of instructors who teach a course in the Spring 2010 semester
 - **select count** (**distinct** *ID*)
from *teaches*
where *semester* = 'Spring' **and** *year* = 2010
- Find the number of tuples in the *course* relation
 - **select count** (*)
from *course*;

집계 함수 – Group By

- Find the average salary of instructors in each department
 - select** *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*;

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

<i>dept_name</i>	<i>avg_salary</i>
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

집계 함수 – Group By (Cont.)

- **select** clause절에 있는 속성들 중에서 집계 함수가 적용되지 않는 속성들은 **group by** 리스트에 반드시 출현해야 함

– /* 여러 질의 */

```
select dept_name, ID, avg (salary)
from instructor
group by dept_name;
```

dept_name으로 그룹화되면 한 그룹 안에
여러 교수 ID가 존재함
→ 그룹의 세부 가지수를 표현할 수 없음
→ 그래서, **select**절에 집계함수(**avg**) 바깥쪽에
있는 속성들이 **group by** 뒤에도 나와야 한다.

집계 함수 – **having** 절

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

주의: **having** 절의 조건은 그룹이 형성된 이후에 적용되는 반면
where 절의 조건은 그룹이 형성되기 전에 적용된다.

널 값과 집계 함수

- 모든 연봉의 집계:

```
select sum (salary )  
from instructor
```

- 합계를 할 때 null 값은 무시함
 - 만약 모두 널 값이면 결과는 0아 이나라 널 값이다.
- count(*)** 제외한 모든 집계 연산은 널 값이 존재하는 튜플을 무시함
- 만약 속성 값들이 모두 널 값이면,
 - count → 0
 - 모든 다른 집계 함수 → null

→ 좀 더 복잡한 SQL 구성요소들에 대한 null 값의 효과는 좀 더 미묘함

3.8 중첩 하위 질의(subquery)

- SQL은 질의들이 중첩되는 것을 허용함
- 하위 질의
 - 질의문 안에 중첩된 또 다른 **select-from-where** 표현
- 하위 질의는 집합 멤버십 확인, 집합 비교, 집합의 갯수를 확인 할 경우 많이 사용됨

중첩 하위 질의 – 집합 멤버십

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
      course_id in (select course_id
                    from section
                    where semester = 'Spring' and year= 2010);
```
- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
      course_id not in (select course_id
                       from section
                       where semester = 'Spring' and year= 2010);
```

중첩 하위 질의 – 집합 비교

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

name	salary	Department
Lee	5000	Biology
Kim	10000	Biology
Park	7000	Computer

- Same query using > **some** clause

```
select name
from instructor
where salary > some (select salary
                        from instructor
                        where dept_name = 'Biology');
```

결과는 ? → (5000, 10000)

- $F <\text{comp}> \text{some } r \Leftrightarrow \exists t \in r \text{ such that } (F <\text{comp}> t)$

Where $<\text{comp}>$ can be: $<, \leq, >, =, \neq$

$$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true} \quad (\text{read: } 5 < \text{some tuple in the relation})$$

$$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 = \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$$

$$(5 \neq \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true (since } 0 \neq 5)$$

- $F \text{ <comp> all } r \Leftrightarrow \forall t \in r (F \text{ <comp> } t)$

$$(5 < \text{all } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \text{all } \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \text{all } \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \text{all } \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

all절의 예

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name  
from instructor  
where salary > all (select salary  
                        from instructor  
                        where dept_name = 'Biology');
```

빈 릴레이션에 대한 검사

- The **exists** 연산자
 - 하위 질의가 공집합이 아니면 **true**를 반환한다.
 - **exists** $r \Leftrightarrow r \neq \emptyset$
 - **not exists** $r \Leftrightarrow r = \emptyset$

- Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester

바깥질의 { **select** *course_id*
from *section* **as** S
where *semester* = 'Fall' **and** *year* = 2009 **and**
exists (select *
하위질의 **from** *section* **as** T
where *semester* = 'Spring' **and** *year* = 2010
and S.*course_id* = T.*course_id*);

연관된 이름

The diagram illustrates the correlation between the outer query and the inner query. A red arrow points from the circled 'S' in the outer query's 'from section as S' clause to the circled 'S' in the inner query's 'and S.course_id = T.course_id' clause. The text '연관된 이름' (Correlated Name) is written in red above the arrow.

- 연관된(Correlated) 하위 질의
- 연관된 이름 혹은 연관된 값

not exists

- Find all students who have taken all courses offered in the Biology department.

select distinct *S.ID, S.name*

from *student as S*

where not exists (**(select** *course_id*
from *course*
where *dept_name* = 'Biology')

생물학과에서 개설된
모든 수업 → X

except

(select *T.course_id*
from *takes as T*
where *S.ID = T.ID));*

학생 S.ID가 수강하는
모든 수업 → Y

참고: $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

중복 튜플의 부재 검사

- **unique** 연산자는 하위 질의의 결과에 중복 튜플이 존재하는지를 검사함
tests whether a subquery has any duplicate tuples in its result.
 - 결과가 없거나 하나인 경우 *true*로 처리함
- Find all courses that were offered at most once in 2009

select *T.course_id*

from *course* **as** *T*

where unique (**select** *R.course_id*
from *section* **as** *R*
where *T.course_id*= *R.course_id*
and *R.year* = 2009);

많아야 한번 → 0 포함?

unique (결과가 없거나 하나인 경우) → true

unique 를 쓰지
않고 표현하면 ?

.....
where 1 >= (**select** count(*R.course_id*)
from *section* **as** *R*
where *T.course_id*= *R.course_id*
and *R.year* = 2009);

참고: 교재(pp. 87)에 오류 있음

from 절의 하위 질의

- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000.

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

select-from-where
결과가 결국 또 다른
relation이기 때문에
가능한 것임

- 또 다른 형태의 동일한 질의

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
      from instructor
      group by dept_name)
as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

from 절 안에 있는 select 구문의 결과를
새로운 relation (즉, dept_avg)로 재명명
→ Oracle에서는 지원되지 않음

- **with**절은 with절을 포함한 질의에서만 이용 가능한 임시 릴레이션
을 정의할 수 있음
- Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
     from department)
```

} with절을 포함한 질의에서만 유효한
임시 relation (*max_budget*) 생성

```
select budget
from department, max_budget
where department.budget = max_budget.value;
```

스칼라 하위 질의

- 스칼라 하위 질의는 단일 값이 기대될 경우 사용할 수 있음

select *dept_name*, ↙ 하위 질의의 결과가 table이 아니라 value인 경우
 (**select** **count**(*)
 from *instructor*
 where *department.dept_name = instructor.dept_name*)
 as *num_instructors*
from *department*;

질의의 의미 ? → 학과별 교수님의 수를 출력

3.9 데이터베이스의 수정

- 릴레이션에서 튜플 제거하기
- 릴레이션에 새로운 튜플 삽입하기
- 릴레이션에 존재하는 기존 튜플의 값을 갱신하기

데이터베이스 수정 - 삭제

- Delete all instructors

delete from *instructor*

- Delete all instructors from the Finance department

delete from *instructor*

where *dept_name* = 'Finance';

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

delete from *instructor*

**where *dept_name* in (select *dept_name*
from *department*
where *building* = 'Watson');**

삭제 (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor  
where salary < (select avg (salary) from instructor);
```

문제: 튜블의 삭제할 때 평균 연봉이 계속 변화한다.

해결책:

1. 먼저, 평균 연봉을 구하고 삭제할 튜플들을 검색한다.
2. 다음 1에서 검색한 모든 튜플을 일괄 삭제한다.

데이터베이스 수정 - 삽입

- Add a new tuple to *course*

insert into *course*

values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

– 혹은,

insert into *course* (*course_id*, *title*, *dept_name*, *credits*)

values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- Add a new tuple to *student* with *tot_creds* set to null

insert into *student*

values ('3003', 'Green', 'Finance', *null*);

삽입 (Cont.)

- Add all instructors to the *student* relation with tot_creds set to 0

insert into student

```
select ID, name, dept_name, 0
from instructor
```

insert 수행전에 select가
먼저 수행되어야 함

- **select-from-where** 문장의 결과는 반드시 릴레이션에 삽입되기 전에
먼저 계산되어야 한다. (만약 그렇지 않으면,

insert into table1 select * from table1

위 문장은 문제가 발생한다.)

만약 select from where 구문을 수행하면서 insert문을 수행하면 어떻게 될까?
즉, select를 해서 하나의 tuple을 뽑고 이를 다시 같은 table에 insert를 하고,
주키 제약조건마저 없다면 어떻게 될까 ?

→ 중복된 tuple들이 계속해서 들어가고 결국 무한 loop에 빠진다.

데이터베이스 수정 - 갱신

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others receive a 5% raise

① { **update instructor**
set salary = salary * 1.03
where salary > 100000;

② → ① 순서로 수행되면 ?

② { **update instructor**
set salary = salary * 1.05
where salary <= 100000;

\$100000 보단 약간 적은 연봉
(즉, 5% 인상하면 \$100000가
넘는 연봉)을 받는 사람들은
다시 3% 더 인상됨 → 본래 의도?

- 갱신의 경우 순서가 중요함
- 혹은 **case**를 사용하면 순서에 덜 민감할 수 있음

조건부 갱신을 위한 **case문**

- case문을 사용한 동일한 갱신

```
update instructor
```

```
set salary = case
```

```
    when salary <= 100000 then salary * 1.05
```

```
    else salary * 1.03
```

```
end
```

스칼라 하위 질의를 이용한 갱신

- 모든 학생들에 대해 `tot_cred`를 재계산하여 갱신할 경우,

update student S

set tot_cred = (~~select~~ **sum(credits)**

from takes natural join course

where S.ID= takes.ID and

takes.grade <> 'F' and

takes.grade is not null);

- 만약, 어떤 과목도 수강하지 않은 학생의 `tot_creds` 을 0으로 세팅하려면 **sum(credits)**을 아래와 같이 수정해야 함

case

when sum(credits) is not null then sum(credits)

else 0

end

어떤 course도 듣지 않은 학생의 `tot_creds`는 null이 아니라 0 이어야 함