

4장. 중급 SQL

이 동 호

데이터베이스 연구실
소프트웨어학부

- 4.1 조인 식
- 4.2 뷰(view)
- 4.3 트랜잭션
- 4.4 무결성 제약조건
- 4.5 SQL의 데이터 타입과 스키마
- 4.6 권한

4.1 조인 식

- **조인 연산**은 두 개의 릴레이션을 하나의 릴레이션으로 합치는 연산이다.
- 조인 연산은 카티셋 곱과 유사하지만 특정한 조건을 만족하는 튜플들만 합친다는 점에서 다르다.
- 조인연산은 일반적으로 **from**절에 표시된다.

조인 연산 – 예제

- *course* 릴레이션

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- *prereq* 릴레이션

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- 참고:
 - *prereq* 릴레이션에는 CS-315가 없음
 - *course* 릴레이션에는 CS-347가 없음

외부 조인

- 서로 match되지 않아 조인 결과에서 빠지는 튜플들을 널 값을 이용하여 보존해 주는 조인
 - 왼쪽 외부 조인
 - 오른쪽 외부 조인
 - 전체 외부 조인

왼쪽 외부 조인

course **natural left outer join** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>

course

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

오른쪽 외부 조인

course **natural right outer join** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

course

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

전체 외부 조인

course **natural full outer join** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

course

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

조인 종류와 조인 조건

- 조인 조건

- 두 개의 릴레이션에서 어떤 튜플들이 매칭되는지를 정의하고 어떤 속성들이 결과 릴레이션에 포함되는지를 명시

- 조인 종류

- 서로 매칭되지 않는 튜플들을 어떻게 처리할지에 따라 구분

기존의 조인을
외부조인과
구분하기 위해서
inner 표시를
한다.

Join types	Join Conditions
inner join left outer join right outer join full outer join	natural on <predicate> using ($A_1, A_1, \dots, A_n$)

조인된 릴레이션 – 예제

- course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

- 위의 조인과 자연 조인의 차이는 무엇인가 ?

- course* **left outer join** *prereq* **on** *course.course_id = prereq.course_id*

on <조건> → 조건에 나오는 속성을
한번 더 표시하는 것 이외에는 동일

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	<i>null</i>	<i>null</i>

외부조인이기 때문에 match되지 않는 튜플을 null을 이용해 표시해 준다.

조인된 릴레이션 - 예제

- course natural right outer join prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

- course full outer join prereq using (course_id)*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

조인 예제

순 번	질 의	예러 유무 (차이점)
1	<code>select * from course natural inner join prereq;</code>	
2	<code>select * from course natural join prereq;</code>	
3	<code>select * from course inner join prereq;</code>	
4	<code>select * from course join prereq;</code>	
5	<code>select * from course inner join prereq using (course_id);</code>	
6	<code>select * from course inner join prereq on course.course_id=prereq.course_id</code>	

참고: 자연 조인에서 자연(natural)은 조인 종류가 아니라 조인의 조건을 의미한다.

4.2 뷰(views)

- 모든 사용자가 데이터베이스에 저장된 릴레이션을 그대로 보는 것이 바람직하지 않을 수 있다.
 - 예를 들어, 학교 직원들은 교수님들의 이름이나 소속 학과는 볼 수 있지만 연봉을 보면
 - 이 경우 SQL은 다음과 같은 질의를 이용해서 가상 릴레이션을 만들 수 있음

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

- 뷰(view)
 - 특정 사용자에게 데이터를 숨길 수 있는 메커니즘을 제공함
 - 즉, 특정 사용자에게만 보이는 가상 릴레이션(**virtual relation**)

뷰 정의

- **create view 문장**

create view v as < query expression >

관리자가 뷰생성 권한을
주어야 가능 “grant create
view to dhlee72”

- v : 뷰 이름
- <query expression> : SQL 문장

- 일단 뷰가 정의되면 뷰 이름(v)은 일반 릴레이션과 동일하게 사용 가능

- 그러나 뷰는 새로운 릴레이션이 만들어지는 것이 아니라 뷰를 정의할 때 사용된 질의문이 저장되어 있다가 사용시 질의문이 대체되는 것임

원 소리? → “v가 사용될 때 v를 정의할 때 사용한 <query expression>이 그대로 대체되어 사용된다”는 뜻!

뷰 예제

- instructors* 릴레이션에서 연봉을 뺀 *faculty* 뷰 생성

create view *faculty* as

select *ID, name, dept_name*
from *instructor*

faculty 라는 view를 정의하는데
사용된 <query expression>

- Find all instructors in the Biology department

select *name*
from *faculty* (select *ID, name, dept_name*
from *instructor*)
where *dept_name* = 'Biology'

- 뷰 정의시 새로운 속성 정의

create view *departments_total_salary*(*dept_name*, *total_salary*) as
select *dept_name*, sum (*salary*)
from *instructor*
group by *dept_name*;

View 정의시 새로운 속성정의 가능!

다른 뷰를 이용한 새로운 뷰 정의

- **create view** *physics_fall_2009* **as**
select *course.course_id, sec_id, building, room_number*
from *course, section*
where *course.course_id = section.course_id*
and *course.dept_name = 'Physics'*
and *section.semester = 'Fall'*
and *section.year = '2009';*

질의 해석 →

2009년 가을 물리학과에서 제공한 모든 수업ID 및 해당 수업에 대한 분반정보(분반ID, 건물이름, 강의실번호)

- **create view** *physics_fall_2009_watson* **as**
select *course_id, room_number*
from *physics_fall_2009*
where *building = 'Watson';*

질의 해석 →

Physics_fall_2009에서 건물이 Watson인 수업ID와 강의실번호

뷰 확장

- 다양한 형태의 뷰 사용

```
create view physics_fall_2009_watson as
select course_id, room_number
from (select course.course_id, building, room_number
from course, section
where course.course_id = section.course_id
      and course.dept_name = 'Physics'
      and section.semester = 'Fall'
      and section.year = '2009')
where building = 'Watson';
```

physics_fall_2009

<정리: view 관리 및 처리>

- view를 정의하면 relation처럼 table이 생성되는 것이 아니라, view를 생성하는 질의 자체를 저장한다.
- view가 사용될 때마다 view 정의에 사용된 질의가 호출된다.

- **실체화 뷰(materialized view)**

- 실체화 view는 질의가 저장되는 것이 아니라, 일반 relation처럼 table형태로 저장됨
- 실체화 view를 정의하는데 사용된 원천 relation이 update되면 반드시 update를 해 주어야 하며, 다양한 방법이 존재함 → 13.5절에서

- *faculty* 뷰에 새로운 튜플을 삽입하는 경우

insert into *faculty* values ('30765', 'Green', 'Music');

위 삽입은 원천 릴레이션(*instructor*)에 아래와 같은 튜플이 삽입되어야 함

('30765', 'Green', 'Music', null)

- View도 update될 수 있으며, 이 경우 DB의 일관성을 위해 원천 relation도 함께 update해 주어야 한다.
- 상기 예에 어떤 문제가 있나? → 별 문제 없어 보임 OK
- 어떤 경우에 문제가 생길 수 있나? → 원천 relation이 2개 이상일 경우

뷰 갱신 (Cont.)

- **create view *instructor_info* as**
select *ID, name, building*
from *instructor, department*
where *instructor.dept_name= department.dept_name*;
- **insert into *instructor_info* values ('69987', 'White', 'Taylor');**

*instructor_info*에 사용된 2개의 원천 relation들은 어떻게 update되어야 하나?

- *instructor* 의 경우 : (69987, white, null, null)
- *department*의 경우 : (null, Taylor, null)
- Taylor 빌딩에 여러 학과들이 있거나 혹은 아무런 학과도 없다면 어떤 식으로 update해야 하나? *dept_name*은 *department* 테이블의 주키임
- 결국, null을 이용한 원천 relation의 update는 한계가 있음
- 대부분 단순한 view인 경우에만 update를 허용함

4.3 트랜잭션

- 데이터베이스에서의 일의 단위 (unit of work)
- 트랜잭션의 성질 (ACID)
 - 원자성(Atomicity) (원자성)
 - 완전히 수행되든지 아니면 전혀 수행되지 않아야 함
 - **commit work(완전한 수행)** 혹은 **rollback work(처음 상태로 되돌아감)**
 - 일관성(Consistency)
 - 트랜잭션의 수행은 데이터베이스의 일관성을 보장해야 함
 - 고립성(Isolation)
 - 다수의 트랜잭션이 동시에 수행되더라도 각 트랜잭션은 다른 트랜잭션의 동시 수행을 모르도록 해야 함
 - 지속성(Durability)
 - 트랜잭션이 성공적으로 수행된 이후 그 변화는 데이터베이스에 영구히 반영되어야 함

무결성 제약조건

- 무결성 제약조건은 데이터베이스의 일관성을 유지하기 위하여 데이터의 결함이 발생하지 못하도록 하는 (제약)조건
 - 어떤 계좌의 잔고는 항상 \$10,000.00보다 커야 한다.
 - 은행원은 시급은 최소 \$4.00이상이어야 한다.
 - 고객은 항상 전화번호가 있다(즉, 고객 테이블에서 전화번호 속성은 널 값이 될 수 없다).

단일 릴레이션에 대한 무결성 제약조건

- **not null**
- **primary key**
- **unique**
- **check (P)**, where P is a predicate

not null and unique 제약조건

- **not null**

- *name*과 *budget*은 **not null**임을 선언

name **varchar(20) not null**

budget **numeric(12,2) not null**

- **unique ($A_1, A_2, \dots, A_m$)**

- 속성 ($A_1, A_2, \dots, A_m$)는 후보키가 될 수 있음을 의미
- 후보키는 널 값이 될 수 있음(반면, 주키는 널 값을 갖을 수 없음)

결국, 어떠한 두개의 튜플도 unique 안에 나열된 모든 속성이 모두 같을 수는 없다. → so, 후보키가 될 수 있음.

- **check (P)** – P는 술어

```
create table section (  
    course_id varchar (8),  
    sec_id varchar (8),  
    semester varchar (6),  
    year numeric (4,0),  
    building varchar (15),  
    room_number varchar (7),  
    time slot id varchar (4),  
    primary key (course_id, sec_id, semester, year),  
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))  
);
```

참조 무결성

- 한 릴레이션에서 출현한 속성(들) 값은 반드시 다른 릴레이션에서도 출현해야 한다.
 - 예: 만약 **Biology**가 **instructor** 릴레이션에서 출현한 학과이름이라면 **department** 릴레이션에서 학과이름이 “**Biology**”인 튜플이 존재해야 한다.
- 외래 키(**foreign key**)
 - A : 속성들의 집합,
 - R 과 S 는 A 를 공통으로 포함하는 두 개의 서로 다른 릴레이션,
 - A 는 S 의 주 키일 때,
 - 만약 R 에 출현하는 A 의 값들이 S 에도 반드시 출현할 때 A 를 R 의 외래 키라고 한다.

참조 무결성에서의 **cascade** 동작

- **create table** *course* (
 course_id **char**(5) **primary key**,
 title **varchar**(20),
 dept_name **varchar**(20) **references** *department*
)
 일반적인 형태: 참조무결성 제약조건 위반시 위반을 유발시킨 동작을 거부함.
- **create table** *course* (
 ...
 dept_name **varchar**(20),
 foreign key (*dept_name*) **references** *department*
 on delete cascade
 on update cascade,
 ...
)
 외래 키절을 이용해 좀 더 구체적인 Action을 명시 → 참조된 릴레이션에서 삭제 및 갱신시 참조하는 릴레이션에도 반영함
- alternative actions to cascade: **set null, set default**

4.5 SQL 데이터 타입과 스키마

- **date**: 년도-월-일
 - 예: **date** '2005-7-27'
- **time**: 시간:분:초
 - 예: **time** '09:00:30' **time** '09:00:30.75'
- **timestamp**: date + time
 - 예: **timestamp** '2005-7-27 09:00:30.75'
- **interval**: 시간 간격
 - 예: **interval** '1' day
 - 날짜와 시간에 대해서 -/+ 연산 가능

4.6 권한

- 데이터베이스 일부를 수정할 수 있는 권한
 - 데이터를 읽을 수 있는 권한
 - 새로운 데이터를 삽입할 수 있는 권한
 - 데이터를 갱신할 수 있는 권한
 - 데이터를 삭제할 수 있는 권한
- 데이터베이스 스키마를 수정할 수 있는 권한
 - 색인을 생성하고 삭제할 수 있는 권한
 - 새로운 테이블을 생성할 수 있는 권한
 - 릴레이션의 속성을 추가하고 삭제할 수 있는 권한
 - 릴레이션을 삭제할 수 있는 권한

SQL을 이용한 권한 부여

- grant 문

grant <privilege list>

(**on** <relation name or view name>) **to** <user list>

- <user list> is:

- 특정 사용자ID
- **Public** : 모든 사용자에게 권한을 줄 경우
- 역할(role)

생략 가능!
grant create view to dhlee72

- 주의

- 뷰에 대한 권한 부여는 원천 릴레이션에 대한 권한 부여를 의미하지 않음
- 권한을 부여하는 사용자는 이미 해당 권한을 소유하고 있어야 함

- Oracle은 200여개의 권한이 있음
- 사용 가능 권한의 종류를 보고 싶다면, **select * from system_privilege_map;**

권한 부여

- **select**: 릴레이션에 대한 읽기 혹은 질의할 수 있는 권한
 - 예제: 사용자 U_1 , U_2 , and U_3 에게 *instructor* 릴레이션에 대한 **select** 권한 부여:

grant select on *instructor* to U_1 , U_2 , U_3

- **insert**: 튜플을 삽입할 수 있는 권한
- **update**: SQL 문장을 이용하여 갱신할 수 있는 권한
- **delete**: 튜플을 삭제할 수 있는 권한
- **all privileges**: 모든 허락된 권한을 줄 때 사용

권한 취소

- **revoke** 문

revoke <privilege-list>

on <relation name or view name> **from** <user-list>

- 예 제:

revoke select on *branch* **from** U_1, U_2, U_3

- 모든 권한을 취소할 때 <privilege-list>에 **all**을 사용
- <user-list> 가 **public**이면 모든 사용자가 권한을 잃음
- 동일한 권한이 서로 다른 사람으로부터 두 번 부여가 된다면,
 - 첫번째 권한이 취소된 이후 다시 권한을 부여 받음
- 취소되는 권한에 종속된 모든 권한들도 동시에 취소됨

역할(Roles)

Roles(역할)이 유용한 경우?

가정) 모든 교수님은 같은 릴레이션의 집합에 대해서 같은 타입의 권한을 가진다.

문제) 새로운 교수님이 임명될 때마다 이러한 권한을 개별적으로 주는 것은 비효율적임

- **create role** instructor;

- 새로운 사용자에게 권한을 부여할 때 유용

- **grant instructor to Amit;**

- 권한을 역할(roles)에 부여하는 것이 가능

- **grant select on takes to instructor;**

- 역할은 다른 역할뿐만 아니라 사용자에게 권한을 부여할 수 있음

create role teaching_assistant

- **grant teaching_assistant to instructor;**

- *instructor*는 *teaching_assistant*의 모든 권한을 상속 받음

- 역할의 체인

- **create role dean;**

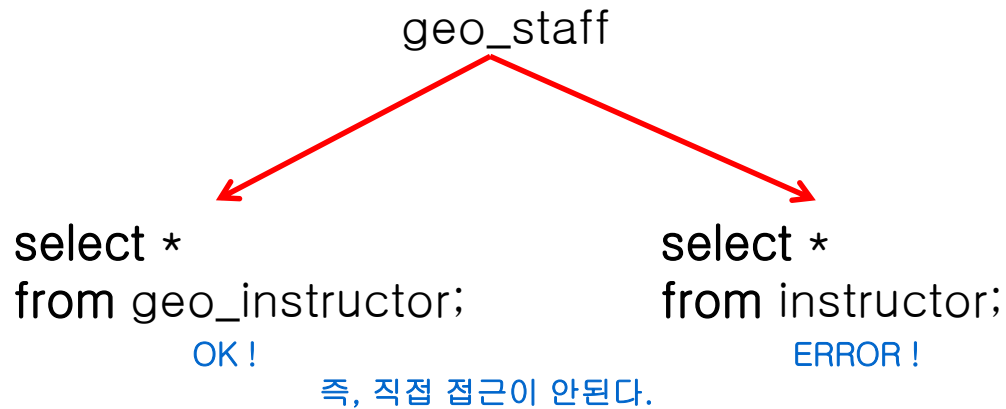
- **grant instructor to dean;**

- **grant dean to Satoshi;**

모든 교수님들에게 일괄적으로 권한을 주는 좋은 방법은?
→ 역할을 만들어 한꺼번에 주는 것

뷰에 대한 권한

- **create view *geo_instructor* as**
(select *
from *instructor*
where *dept_name* = 'Geology');
- **grant select on *geo_instructor* to *geo_staff***
- *geo_staff* 가 아래의 질의를 수행할 경우, 그 결과를 볼 수 있나?
 - **select ***
from *geo_instructor*; } → 당연히 볼 수 있다(select 권한이 있음으로..)
- 만약,
 - 뷰 생성자가 *instructor*에 대한 **select** 권한이 없다면 ?
 → view를 생성한 사람은 이미 *instructor*에 대한 select 권한이 있다.
 없다면 view를 처음부터 만들지도 못했겠지!
 - *geo_staff* 가 *instructor*에 대한 권한이 없다면?
 → SQL 질의 처리기는 해당 질의를 처리하기 전에 권한 검사를 먼저 수행하여
 권한이 있는 경우 정상 처리하지만, 없으면 거부하게 된다.



- 뷰에 대한 권한 부여는 원천 릴레이션에 대한 권한 부여를 암시하지 않는다.

[Oracle에서 실습해 보기]

- 새로운 사용자를 하나 더 만들어서 앞의 예를 실험해 보자!
- 주의) 다른 사용자의 소유의 객체를 접근하기 위해서는 “사용자.객체명” 형태로 접근해야 함
- user1과 user2가 있다고 할 때, user2가 user1이 만든 객체를 접근할 때 아래와 같이 사용해야 함

```

select *
from user1.instructor
  
```