Chapter 4 Review (Intermediate SQL)

4.1 Join Expressions

명령	문법	설명
일반적인 조인 (Inner Join)	SELECT <속성들> FROM 테이블1, 테이블2 WHERE <조인조건> AND <검색조건>	SQL 문에서는 주로 동등조인을 사용. 두 가지 문법 중 하나를 사용할 수 있음. 자연조인은 일반적으로 Inner Join 임
	SELECT <속성들> FROM 테이블1 INNER JOIN 테이블2 ON <조인조건> WHERE <검색조건> SELECT <속성들> FROM 테이블1 NATURAL JOIN 테이블2 WHERE <검색조건>	
외부조인 (Outer Join)	SELECT <속성들> FROM 테이블1 {LEFT |RIGHT |FULL [OUTER]} JOIN 테이블2 ON <조인조건> WHERE <검색조건> SELECT <속성들> FROM 테이블1 NATURAL {LEFT |RIGHT |FULL [OUTER]} JOIN 테이블2 WHERE <검색조건>	외부조인은 FROM 절에 조인 종류를 적고 ON을 이용하여 조인조건을 명시함. 주의: 앞에 NATURAL을 붙이는것이랑 안붙이는것이랑 결과값이 다름

Outer Join

An extension of the join operation that avoids loss of information.

Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.

Uses *null* values.

외부조인 - 서로 match되지 않아 조인 결과에서 빠진 투플들을 null을 이용해 보존하는 조인

Left outer join:

왼쪽을 무조건 채우는 것으로 오른쪽 테이블에 null이 존재

Include the left tuple even if there's no match

Right outer join:

오른쪽을 무조건 채우는 것으로 왼쪽 테이블에 null이 존재

Include the right tuple even if there's no match

Full outer join:

Include the both left and right tuples even if there's no match

Left Outer Join

*select * from course* **natural left outer join** *prereq*

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

course

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

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Joined Relations – Examples

*Select * from course* **left outer join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
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	*null*	*null*

외부조인이기 때문에 match되지 않는 튜플을 null을 이용해 표시해 준다.

*select * from course* **natural right outer join** *prereq*

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

Full Outer Join

*select * from course* **natural full outer join** *prereq*

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

course

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

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Joined Relations – Examples

course **full outer join** *prereq* **using** (*course_id*)

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

- **join – using은 앞에 NATURAL을 붙인것 과 같음**
- **SQL에서 어떤 attribute가 같아져야하는지 명시적으로 표시하는 구문**

4.2 Views

A **view** provides a mechanism to hide certain data from the view of certain users.

Any relation that is not of the conceptual model but is made visible to a user as a "**virtual relation**" is called a **view**.

- **하나 이상의 테이블을 합하여 만든 가상의 릴레이션(테이블)**
- **특정 데이터를 숨기는 형태로 가상의 릴레이션을 정의하여 실제 릴레이션처럼 사용 가능**

A view is defined using the **create view** statement which has the form

create view *v* **as** < query expression >

where <query expression> is any legal SQL expression. The view name is represented by *v.*

View Definition

View definition is not the same as creating a new relation by evaluating the query expression

Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

→ *View*가 사용될 때 *view*를 정의할 때 사용한 <query expression>이 그대로 대체되어 사용된다"는 뜻!

create view *vw_Customer* **as**
select *
from *Customer*
where *address LIKE '%대한민국%'*

*Vw_Customer*라는 view를 정의하는데 사용된 <query expression>

select * **from** vw_*Customer*

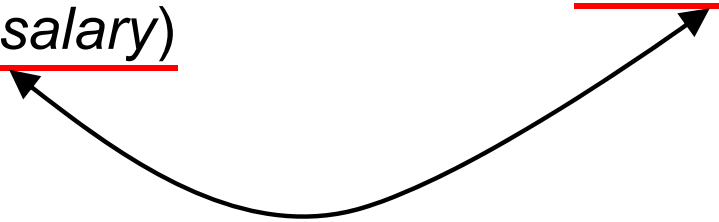
(**select** * **from** *customer*
where *address LIKE '%대한민국%'*)

Cust_ID	NAME	ADDRESS	PHONE
2	김연아	대한민국 서울	000-6000-0001
3	장미란	대한민국 강원도	000-700-0001

Example Views

Create a view of department salary totals

```
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!

<정리: view 관리 및 처리>

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

Update of a View

Add a new tuple to *faculty* view which we defined earlier

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

This insertion must be represented by the insertion of the tuple

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

into the *instructor* relation

- 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

Some Updates cannot be Translated

create view *instructor_info* **as**
select *ID*, *name*, *building*
from *instructor*, *department*
where *instructor.dept_name= department.dept_name*;

Instructor

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Taylor	75000
98345	Kim	Taylor	80000

department

dept_name	building	budget
Biology	Watson	90000
Comp. sci	Taylor	100000
Elec. Eng.	Taylor	85000

Instructor_info

ID	name	building
76766	Crick	Watson
45565	Katz	Taylor
98345	Kim	Taylor

insert into *instructor_info* **values** ('69987', 'White', 'Taylor');
- instructor_info에 사용된 2개의 원천 relation들은 어떻게 update되어야 하나?

Some Updates cannot be Translated

insert into *instructor_info* **values** ('69987', 'White', 'Taylor');

- instructor_info에 사용된 2개의 원천 relation들은 어떻게 update되어야 하나?

Instructor

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
69987	White	null	null

department

dept_name	building	budget
Biology	Watson	90000
Comp. sci	Taylor	100000
Elec. Eng.	Taylor	85000
null	Taylor	null

Instructor_info

ID	name	Building
76766	Crick	Watson
45565	Katz	Taylor
98345	Kim	Taylor

오류발생!!!

Null = Null은 Unknown임!
즉 false임으로 insert한 값이
Join 테이블에 나올 수 없음

Some Updates cannot be Translated

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');

- which department, if multiple departments in Taylor?
- what if no department is in Taylor?

instructor_info에 사용된 2개의 원천 relation들은 어떻게 update되어야 하나?

- instructor 의 경우 : (69987, white, null, null)
- department의 경우 : (null, Taylor, null)

즉, 실제로 이값이 입력되도 공통키값이 null이기 때문에 view로는 볼수 없음

- Taylor 빌딩에 여러 학과들이 있거나 혹은 아무런 학과도 없다면 어떤 식으로 update해야 하나?

→ 결국, null을 이용한 원천 relation의 update는 한계가 있음
→ 대부분 단순한 view인 경우에만 update를 허용함

Some Updates cannot be Translated

Most SQL implementations allow updates only on simple views

- The **from** clause has only one database relation.
- The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
- Any attribute not listed in the **select** clause can be set to null

  primarykey나 not null 속성이 지정되지 않은 경우 simple view가 아님

  (앞의 예에서 dept_name 같은 속성 → null이 되면 안됨!!)
- The query does not have a **group** by or **having** clause.

Another Problem of View Update

create view *history_instructors* **as**
  select *
  from *instructor*
  where *dept_name*= 'History';

Update가 가능한 단순 view

What happens if we insert ('25566', 'Brown', 'Biology', 100000) into *history_instructors?*

- 갱신 가능한 단순 view이어서 상기 insert 문이 수행되면, 이 투플은 instructor에 삽입은 되지만, 정작 history_instructor 뷰에는 나타나지 않음
- history_instructor 뷰를 대상으로 했는데, 나타나지 않는다면 바람직하지 않음
- 이 경우**, with check option** 절을 추가하여 갱신을 제한할 수 있다.
  (즉, 뷰에 삽입되는 투플이 where절을 만족하지 못하면 삽입될 수 없도록
  뷰 정의시 where절 다음에 "with check option" 만 추가하면 됨)

4.3 Transactions

Properties of the transactions (ACID)

Atomicity (원자성) 각 원소가 무엇인지 역으로 묻거나 해당 내용이 뭔 주제인지 확인하는 시험문제 나옴

- Either fully executed or rolled back as if it never occurred
- All transactions are ended by **commit work** or **rollback work**

Consistency (일관성)

- Execution of a transaction preserves the consistency of the database 트랜젝션의 수행의 관계없이 db의 데이터는 항상 일정

Isolation (고립성? 독립성)

- Even though multiple transactions may execute concurrently, the system guarantees that, each transaction is unaware of other transactions executing concurrently in the system

Durability (내구성)

- After a transaction completes successfully, the changes it has made to the database persist, even of there are system failures

  저장도중에 에러가 나는경우 계속 기록되는 로그데이터에서 값을 복구함

Integrity Constraints

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

- A checking account must have a balance greater than $10,000.00
- A salary of a bank employee must be at least $4.00 per an hour
- A customer must have a (non-null) phone number

not null

primary key

unique

check (P), where P is a predicate

Not Null and Unique Constraints

unique (A_1, A_2, …, A_m)

The unique specification states that the attributes *A*1, *A*2, … *A*m form a candidate key.

Candidate keys are permitted to be null (in contrast to primary keys).

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

여기에서 이야기하는 UNIQUE는, 3장에서 서브쿼리를 배울 때 나온 UNIQUE와는 다름

- 3장에서 서브쿼리를 배울 때 나온 UNIQUE는 중복 튜플을 가지는 여부를 테스트할 때 사용됨
- 이 때 Unique는 질의문에서 항상 WHERE절 다음에 사용되며, 서브쿼리가 나옴

```
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);
```

Not Null and Unique Constraints

unique (A_1, A_2, …, A_m) 시험문제에는 나오지 않음

The unique specification states that the attributes *A*1, *A*2, … *A*m form a candidate key.

Candidate keys are permitted to be null (in contrast to primary keys).

```
CREATE TABLE 테이블 명
 (컬럼명 1 데이터타입 UNIQUE,
  컬럼명 2 데이터타입,
  컬러명 3 데이터타입,
  컬럼명 4 데이터타입, UNIQUE CONSTRANITS 제약명  UNIQUE (컬럼2, 컬럼3));
```

1. **Unique는 Create Table에서 사용되며, Unique가 사용된 컬럼은 해당테이블에서 존재하는 값이 유일해야 함**
2. **Null 값에 대해서는 Unique 제약이 적용되지 않음 (Null값은 데이터로 인식하지 않기 때문에 해당 컬럼에 Null 데이터행이 여러 개 존재 가능)**
3. **테이블을 만들 때 CONSTRRAINTS 제약명 (컬럼2, 컬럼3)과 같이 CONSTRAINTS의 이름 지정 가능 (단 UNIQUE 제약으로 들어가는 컬럼들은 그들의 조합이 유일해야함!)**
   ***각 컬럼의 데이터의 유일함이 아니라 조합이 유일해야 함**
   e.g. 컬럼 2와 컬럼 3에 각각 1,2 그리고 1,3의 데이터는 존재 가능!!
   1이라는 값이 동일하더라도 조합이 다르기 때문에 오류를 일으키지 않음

The check clause

check (P)

where P is a predicate

Example: ensure that semester is one of fall, winter, spring or summer:

```
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'))
);
```

Referential Integrity

Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S.

참조 무결성 제약조건은 두 릴레이션간 참조관계에 따른 제약조건 임

R (학생 릴레이션)

학번	이름	학과코드
501	박지성	1001
401	김연아	2001
402	장미란	2001
502	추신수	1001

FK

S (학과 릴레이션)

학과코드	학과명
1001	컴퓨터공학과
2001	체육학과

PK

참조

학과코드
- **R(학생 릴레이션)의 외래키**
- **S(학과 릴레이션)의 주키**

A (학과코드) = {1001, 2001}

Cascading Actions in Referential Integrity

```
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 Action을 거부함.
- Department에 없는 dept_name을 course에 삽입
- Department를 course보다 먼저 삭제하는 경우 등

```
create table course (
    …
    dept_name varchar(20),
    foreign key (dept_name) references department
            on delete cascade
            on update cascade,
    . . .
)
```

foreign key 절을 이용해 좀 더 구체적인 Action을 명시 → 참조된 릴레이션에서 삭제 및 갱신시 참조하는 릴레이션에도 반영함

alternative actions to cascade: **set null**, **set default**

Index Creation

인덱스 고려사항이나 정의에 대한 문제 1문제나옴

create index *studentID_index* **on** *student*(*ID*)

Indices are data structures used to speed up access to records with specified values for index attributes

e.g. **select * from** *student* **where** *ID* = '12345'

can be executed by using the index to find the required record, without looking at all records of *student*

인덱스를 생성하였다고 데이터검색이 무조건 빨라지는 것은 아님

데이터양이 별로 없거나 데이터 값이 몇종류 안되면 없는것이 더 빠름

인덱스 생성 고려사항

1. **인덱스는 where절에서 자주 사용되는 속성이어야 함**
2. **인덱스는 조인에 자주 사용되는 속성이어야 함**
3. **단일 테이블에 인덱스가 많으면 속도가 느려질 수 있음** (테이블 당 4~5개 권장)
4. **속성이 가공되는 경우 사용하면 안됨** 집계함수나 group by사용할때의 경우

User-Defined Types vs. Domains

create type construct in SQL creates user-defined type

create type *Dollars* **as numeric (12,2) final**

create table *department*
(*dept_name* **varchar** (20),
building **varchar** (15),
budget *Dollars*);

- SQL99에서 정의
- 현재는 아무런 의미 없음
- 어떤 시스템에선 생략가능

create domain construct in SQL-92 creates user-defined domain types

create domain *person_name* **char**(20) **not null**

* 도메인과 타입의 차이
1) 도메인은 not null과 같은 제약조건을 가지며, default 값을 가질 수 있다.
2) 도메인은 강력한 타입이 아님. 호환 가능할 수도 있음

Authorization Specification in SQL

The **grant** statement is used to confer authorization

grant <privilege list>

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

생략 가능!
grant create view to dhlee72

<user list> is:

a user-id

public, which allows all valid users the privilege granted

A role (more on this later)

Granting a privilege on a view does not imply granting any privileges on the underlying relations.

The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

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

Privileges in SQL

select: allows read access to relation,or the ability to query using the view

Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

grant select on *instructor* **to** U_1, U_2, U_3 이 구문 적을줄 알아야함

insert: the ability to insert tuples

update: the ability to update using the SQL update statement

delete: the ability to delete tuples.

all privileges: used as a short form for all the allowable privileges

Select, insert, update, delete 등의 권한은 개별적으로 부여해야 하는 권한임

상위권한과 하위권한은 Role(역할)을 가지고 권한을 부여할 때 발생

Revoking Authorization in SQL

The **revoke** statement is used to revoke authorization.

revoke <privilege-list>

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

Example:

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

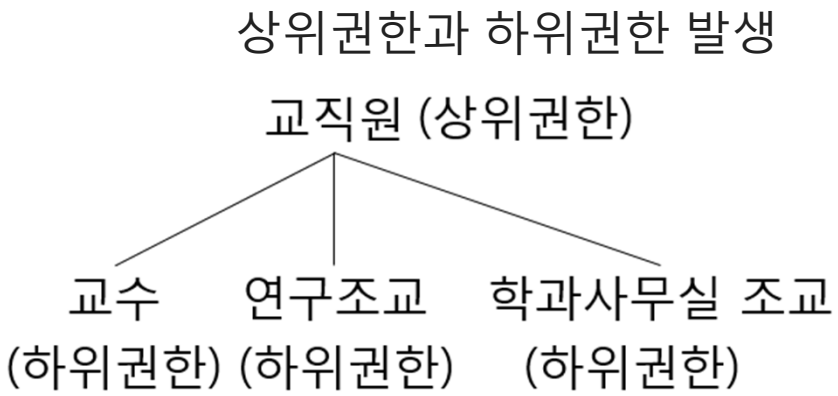
<privilege-list> may be **all** to revoke all privileges the revokee may hold.

If <user-list> includes **public,** all users lose the privilege except those granted it explicitly.

If the same privilege was granted twice to the same user by different grantors, the user may retain the privilege after the revocation.

All privileges that depend on the privilege being revoked are also revoked.

Roles



create role instructor;

grant *instructor* **to** Amit**;**

Privileges can be granted to roles:

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

Roles can be granted to users, as well as to other roles

create role *teaching_assistant*

grant *teaching_assistant* **to** *instructor*;

- *Instructor* inherits all privileges of *teaching_assistant*

Chain of roles

create role *dean*;

grant *instructor* **to** *dean*;

grant *dean* **to** Satoshi;

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

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

Authorization on Views

권한중에 가장 중요

create view *geo_instructor* **as**
(**select** *
from *instructor*
where *dept_name* = 'Geology');

grant select on *geo_instructor* **to** *geo_staff*

Suppose that a *geo_staff* member issues

select *
from *geo_instructor*;

geo_staff가 과연 이 질의의 결과를 볼 수 있을까?
→ 당연히 볼 수 있다(select 권한이 있음으로..)

What if

creator of view did not have some permissions on *instructor?*
→ view를 생성한 사람은 이미 instructor에 대한 select 권한이 있다.
없다면 view를 처음부터 만들지도 못했겠지!

geo_staff does not have permissions on *instructor?*
→ SQL 질의 처리기는 해당 질의를 처리하기 전에 권한 검사를 먼저 수행하여 권한이 있는 경우 정상 처리하지만, 없으면 거부하게 된다.