Chapter 4: Intermediate SQL

Revision by Gun-Woo Kim

Dept. of Computer Science and Engineering

Hanyang University

Database System Concepts, 6th Ed.

Contents

4.1 Join Expressions

Join operations take two relations and return as a result another relation.

카테이션 프로덕트를 이용해서 조인

A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition).

It also specifies the attributes that are present in the result of the join

The join operations are typically used as subquery expressions in the **from** clause

Join operations – Example

Relation *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

Relation *prereq*

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

Observe that

prereq information is missing for CS-315 and

course information is missing for CS-347

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: 왼쪽 데이터만 보존

Include the left tuple even if there's no match

Right outer join: 오른쪽 테이블 데이터만 보존

Include the right tuple even if there's no match

Full outer join: 오른쪽 왼쪽테이블데이터 보존

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

Right Outer Join

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

natural 키워드가 없으면 뒤에 using이라던가 on키워드로 조건을 지정해줘야함

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

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

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

Join Types and Conditions

Join operations take two relations and return as a result another relation.

These additional operations are typically used as subquery expressions in the **from** clause

Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join.

Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

기존의 조인을 외부조인과 구분하기 위해서 inner 표시를 한다.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Joined Relations – Examples

*Select * from course* **inner 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

What is the difference between the above, and a natural join?

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

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

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.

Joined Relations – Examples

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

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

순번	질 의	에러 유무 (차이점)
1	**select** * **from** course **natural inner join** prereq;	x
2	**select** * **from** course **natural join** prereq;	x
3	**select** * **from** course **inner join** prereq;	조건없음
4	**select** * **from** course **join** prereq;	조건없음
5	**select** * **from** course **inner join** prereq using (course_id);	x
6	**select** * **from** course **inner join** prereq on course.course_id=prereq.course_id;	x

[NOTICE]
natural join에서 natural은 operation 종류가 아니라 join의 조건을 의미한다.

4.2 Views

In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)

Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

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

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**.

View Definition

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

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

관리자가 뷰생성 권한을 주어야 가능 "grant create view to gwkim81"

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

Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

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.

뭔 소리? → "*v*가 사용될 때 *v*를 정의할 때 사용한 <query expression>이 그대로 대체되어 사용된다"는 뜻!

Example Views

A view of instructors without their salary

```
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
where dept_name = 'Biology'
```

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

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!

Views Defined Using Other Views

```
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';
```

질의해석 → 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';
```

질의해석 → Physics_fall_2009에서 건물이 Watson인 수업ID와 강의실번호

View Expansion

Expand use of a view in a query/another view

```
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.

Materialized Views

Materializing a view:

Create a physical table containing all the tuples in the result of the query defining the view

If relations used in the query are updated, the materialized view result becomes out of date

Need to **maintain** the view, by updating the view whenever the underlying relations are updated.

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

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*;

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)
- 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

  (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

Unit of work

Properties of the transactions (ACID)

Atomicity (원자성)

- 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

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

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

Integrity Constraints on a Single Relation

not null

primary key

unique

check (P), where P is a predicate

Not Null and Unique Constraints

not null

Declare *name* and *budget* to be **not null**

name **varchar**(20) **not null**
budget **numeric**(12,2) **not null**

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.

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

Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology".

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.

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.

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**

4.5 SQL Data Types and Schema – Date and Time Types in SQL

date: Dates, containing a (4 digit) year, month and date

Example: **date** '2005-7-27'

time: Time of day, in hours, minutes and seconds.

Example: **time** '09:00:30' **time** '09:00:30.75'

timestamp: date plus time of day

Example: **timestamp** '2005-7-27 09:00:30.75'

interval: period of time

Example: interval '1' day

Subtracting a date/time/timestamp value from another gives an interval value

Interval values can be added to date/time/timestamp values

Index Creation

create table *student*
(*ID* **varchar** (5),
name **varchar** (20) **not null**,
dept_name **varchar** (20),
tot_cred **numeric** (3,0) **default** 0,
primary key (*ID*))

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*

More on indices in Chapter 11

Large-Object Types

Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:

blob: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)

clob: character large object -- object is a large collection of character data

When a query returns a large object, a pointer is returned rather than the large object itself.

User-Defined Types

create type construct in SQL creates user-defined type

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

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

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

Domains

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

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

Types and domains are similar. Domains can have constraints, such as **not null**, specified on them.

create domain *degree_level* **varchar**(10)
constraint *degree_level_test*
check (**value in** ('Bachelors', 'Masters', 'Doctorate'));

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

4.6 Authorization

Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema

- **Index** - allows creation and deletion of indices.
- **Resources** - allows creation of new relations.
- **Alteration** - allows addition or deletion of attributes in a relation.
- **Drop** - allows deletion of relations.

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

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

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

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

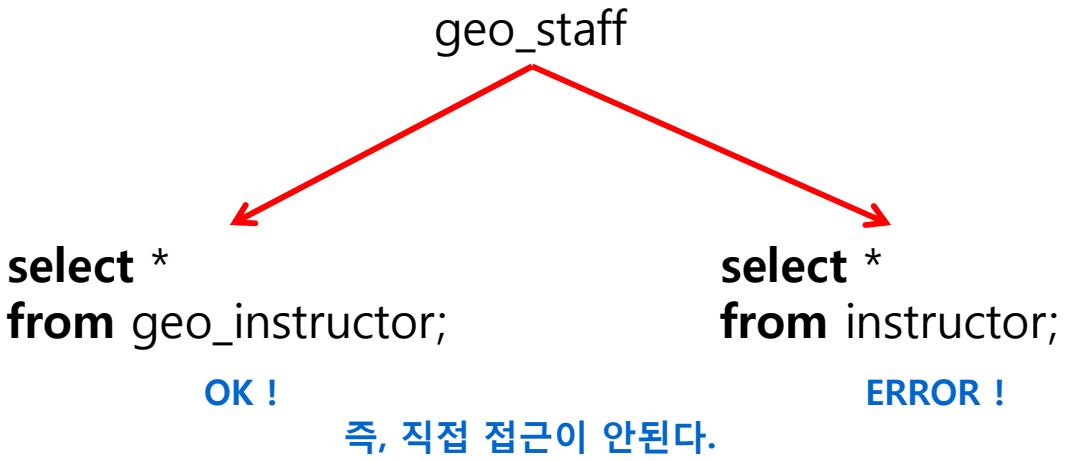
select *
from *geo_instructor*;

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.



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

[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
```