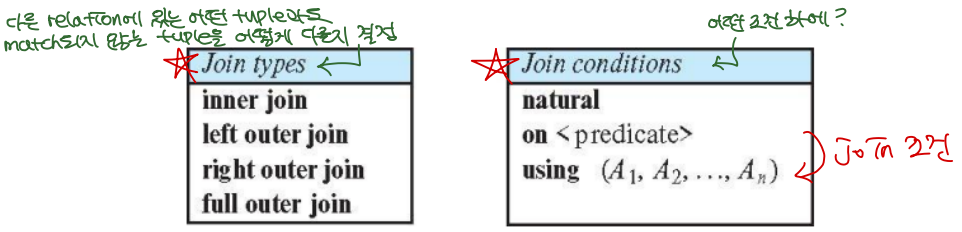
7,8,9. Intermediate SQL

▼ Join Expressions : 두개의 table 비교 ⇒ 조건에 해당되는 tuple합친것.



▼ join operations

- 두 개의 realtion을 다른 하나의 realtion으로 return
- 연관된 column으로 두 개 이상의 table의 row를 합침
- from문 내에서 subquery expression으로 사용

1. **Natural Join**

```
select A1, A2, … An
from r1 natural join r2 ... natural join rn
where P ;
```

Attribute / Relation

- 두 개의 table 비교 → 같은 attribute 내의 value 비교 ⇒ 같으면 합침

```
//y라는 교수님이 가르친 수업을 들은 모든 학생 list up
//기존 query
select name, course_id
from students, takes
where student.ID = takes.ID;

//natural join ver.
select name, course_id
from student natural join takes; //relation 사이에 작성
```

Where문 사용 X / 두 relation의 같은 attribute 비교

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Student relation

같은 attribute끼리 join.

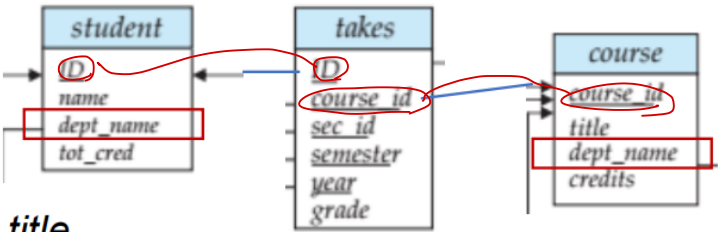
ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	null

takes relation

student natural join takes

ID	name	dept_name	tot_cred	course_id	sec_id	semester	year	grade
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	null

- Danger in Natural Join
  - 같은 이름을 가진 unrelated attribute
    - 만약 같은 이름의 두 column → 자동적으로 Natural join
      ⇒ 겹치는 attribute가 있는지 체크할 것, 연관이 없으면 natural join 사용 X
      +) join을 해도 attribute가 다른 table의 소유 X
  - example



Put, dept_name도 공통으로 들어있는 attribute ⇒ 합쳐 이중으로 출력.

```
//학생들이 들었던 수업의 이름과 학생 이름
//incorrect
select name, title
from student natural join takes natural join course;

//correct
```

→ dept_name도 공통
⇒ 같이 natural join 됨
→ 다른 학과 수업을 들은 사람은 check X

→ 3개 이상 join할 때 더 유의할 것

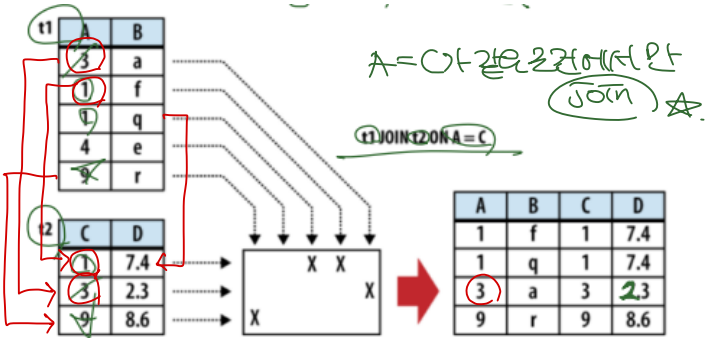
```
select name, title
from student natural join takes, course //공통된 놈은 뭘해도 상관 없도록
where takes.course_id = course.course_id;
```

- using construct 사용
  - 겹치는 column 무시하고 특정 attribute만 고려하도록

```
select name, title
from (student natural join takes) natural join course
using (course_id)
```

2. **Inner Join**

- join condition



  - *join - on* : 조건 쓸 때 사용
  - example

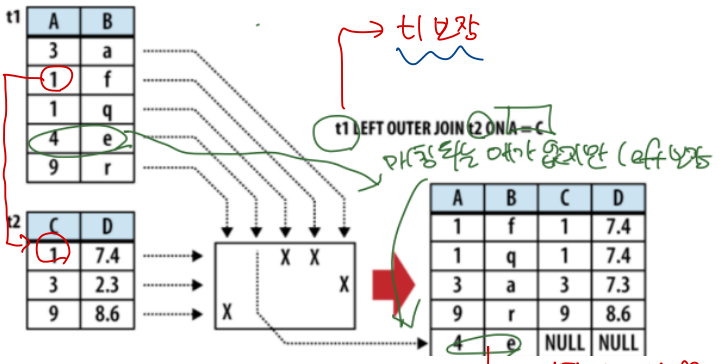
```
select *
from student join takes on student_ID = takes_ID

//equivalent
select *
from student , takes
where student_ID = takes_ID
```

3. **Outer Join**

정보의 손실을 막는 join operation

- compute the join → join의 결과 + 다른 relation과 매치되지 않는 realtion
- null value 사용
- example

*left outer join



예제 table

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

- **left outer join**
  1. relation algebra : course ⟕ prereq
  2. example

```
course natural left outer join prereq
//뒤에 on으로 조건이 붙을 수도 있음
```

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*

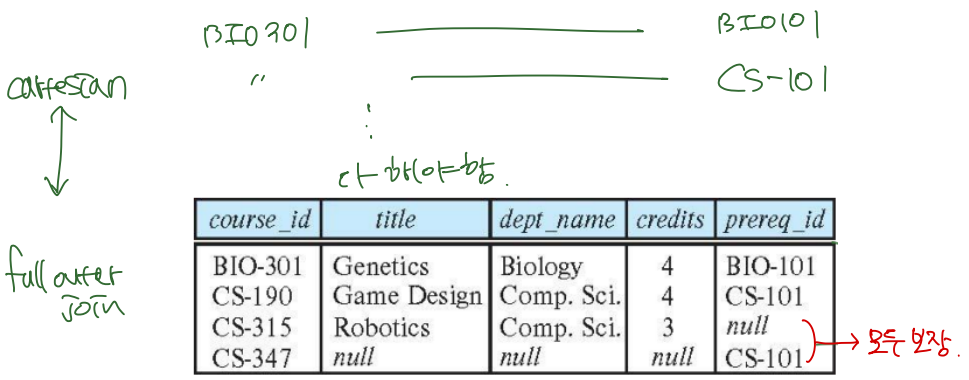
- **right outer join**
  1. relation algebra : course ⟖ prereq
  2. example

```
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 other join (≠ Cartesian product)**
  1. relation algebra : course ⟗ prereq
  2. example
     a. 존재하지 않는 tuple은 null 값으로 채워 넣음 → data 보장

```
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

→ 모두 보장.

▼ Join Relations

- example1

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

보장.

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

→ 매칭안되어도 추가

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

→ course-id 사용해서 매칭 안 되어도 추가

- example2

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

course **inner join** *prereq* **on**
course.course_id = prereq.course_id ← Join 조건 작성

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

보장

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*

→ 매칭 안된것도 추가

▼ Views

▼ Definition and use

ex) 교수의 정보는 알아야 할 필요가 있지만 연봉은 알 필요가 없을 때

→ salary attribute 제외하고 임시적인 table 생성해서 user에게 제공

view: DROP table 할 때까지 유지됨.
with: 한 번 실행할 query이기에 그 query를 내에서만 실행 ☆

```
create view faculty as
select ID, name, dept_name from instructor
```

→ 임시적인 table 생성

```
//만든 table에서 biology 학과의 모든 교수를 찾음
select name
from faculty where dept_name = 'Biology'
```

ex2) 학과별 연봉 총합 구할 때

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

▼ Defined Using Other Views

- 하나의 view : 다른 view 에서 또다른 view를 만들기 위해 사용 가능
- **depend directly** : V2가 만들어질 때 V1 사용함 → V1은 V2에 직접적

```
create view physics_fall_2017_watson as
select course_id, room_number
from physics_fall_2017 //view를 가지고 또 다른 view를 만들 수 있음
where building= 'Watson';
```

- **depend on** : V1가 V2에 직접적 or V1에서 V2로 path of dependencies가 있을 때

  → V1은 V2에 의존적

```
create view physics_fall_2017 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 = '2017';
```

+) view가 삭제되는 시점 → view를 삭제하는 명령어 실행될 때

⇒ 임시적일수도 있고, 아닐수도 있음(table 자체를 storage로 저장 x)

⇒ query문이 시행될 때 생성됨

▼ View Expansion

다른 view의 관점에서 정의된 view의 의미를 define하는 방법

→ view *V1*이 view relation이 포함된 expression *e1*에 의해 정의되었다고 가정

→ **view expansion**

repeat
Find any view relation *vi* in *e1*
Replace the view relation *vi* by the expression defining *vi*
until no more view relations are present in *e1*

```
create view physics_fall_2017_watson as
select course_id, room_number
from physics_fall_2017
where building= 'Watson'
```

```
>> expand를 펼쳐쓰면

create view physics_fall_2017_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 = '2017')
where building= 'Watson'
```

▼ Materialized Views

- 특정 database system : physically하게 저장되는 view relation
  - physical copy : view가 define되었을 때 생성됨
    - **materialized view**라고 불림 ⇒ 독립되어서 저장, 최신화 x
- 광장히 큰 view : join할 때 너무 복잡 ⇒ 저장공간 사용 ⇒ 연산이 많아도 수행 속도 Good.
- 만약 query 내부에 있는 relation이 update된다면 materialized view는 out of date
  - table update할 때 view도 update해줘야 함

    → update를 자주 하는 table의 경우 효율이 좋지 않을 수 있음

▼ Update of a View

1. *add*
   - 우리가 이미 define 해놨던 view에 ***faculty***라는 새로운 tuple add

```
Insert into faculty
values ('30765', 'Green', 'Music')
```

   - insertion : instructor relation에 insertion하는 경우 → salary values 가져야 함
2. *Reject insert* ⇒ 하지마는 것.
   - error : tuple 이름을 define 해줘야 함

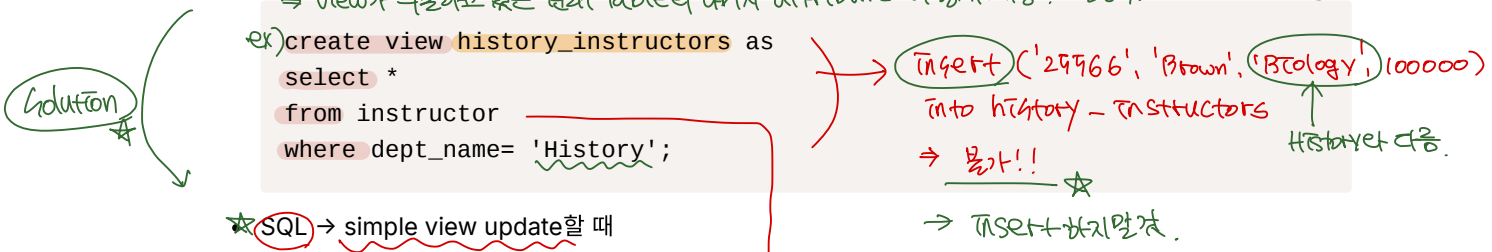
```
Inset the_tuple
('30765', 'Green', 'Music', null)
```

   - some example

```
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');
```

- Taylor 빌딩 안에 여러 개의 department 존재 가능?
- Taylor 빌딩 안에 아무 department도 존재 x?

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



SQL → simple view update할 때

1) *from* : 하나의 data base relation만 작성
2) *select* : relation의 attribute name만 포함
   - expression, aggregate, distinct specification 모두 포함 x
   - select에 작성된 어떤 attribute도 null x
3) *group by, having* : 작성 X

▼ Transactions

database system의 실행 단위(consists of a sequence of query and/or update statements)

- SQL이 실행되었을 때 명시적으로 시작됨
- transaction statements
  1. Commit work : transaction에 의한 update가 영구적으로 실행
  2. Rollback work : transaction에 의한 모든 update가 undone
     a. 아무 일도 없던 것처럼 돌아가야 함(insert 문이 정상적으로 끝나지 x)

atomic transaction

- 완전히 실행 혹은 아예 실행된적 X
- isolation from concurrent transactions
  - 사람들은 최대한 많은 일을 빠르게 처리하고 싶어 함.
    - 서로 다른 transaction이 같은 data에 접근/작성하는 일이 발생 가능

▼ Integrity Constraints

database의 변화가 data inconsistency로 data가 변경되지 않도록 보장함으로써 database 보호

ex) 은행원의 연봉이 시간 당 4불은 되어야 함.

- constraints on a single relation
  1. *not null*
     ```
     name varchar(20) not null
     budget numeric(12,2) not null
     (attribute)
     ```
  2. primary key
  3. *unique*
     a. candidate key인 attribute 사용

b. candidate key → null 값 포함 (↔ primary key)

```
unique (A1, A2, …, Am)
            //attribute 이름 나열 => candidate key
```

4. *check*

a. relation 내부의 모든 tuple에서 predicate P

```
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), //4개의 attribute가 primary key
check (semester in ('Fall', 'Winter', 'Spring', 'Summer')))
```

b. Complex Check conditions

- *check* : subquery를 가지고 있을 수 있는 임의의 표현법이 될 수 있음

```
check (time_slot_id in (select time_slot_id from time_slot))
```

- ***section*** relation 내부에 있는 각 tuple의 ***time_slot_id***가 ***time_slot*** relation에 있는 ***time_slot***의 identifier
- condition : tuple이 section 내부에서 inserted or modified 될 때 뿐만 아니라 time_slot relation이 변할 때도 checked되어야 함

- **Referential Integrity**
  - 하나의 relation에 나왔다면 다른 relation에도 반드시 나와야 하는 값을 보장

    ex) instructor relation : “Biology”라는 dep_name attribute 값이 있음

    → department relation : “Biology”라는 dep_name attribute 값을 가진 tuple이 있어야 함

  - definiiton
    - A가 다른 relation S1의 primary key라면 A는 S의 primary key는 아니지만 foreign key
    - A가 R의 foreign key 중 하나의 값이라면 S에 반드시 나와야 함
  - default → foreign key가 referenced table의 primary-key attribute들을 나타냄
  - SQL : 관련된 relation의 attribute들을 나열하는 것을 제공함

```
foreign key (dept_name) references department (dept_name)
//attribute 이름을 직접적으로 명시함.
```

- cascading actions in **Referential Integrity**
  - Referential Integrity constraint 위반했을 때
    ① normal procedure : violation 일으키는 action 거부
    ② set null, set default 사용하기도 함
    ③ *delete* or *update* ⇒ cascade

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

  - during transaction → violation 발생, How ?

```
create table person (
    ID char(10),
    name char(40),
    mother char(10),
    father char(10),
primary key ID,
foreign key father references person,
foreign key mother references person) //table이 자기 자신을 걸고 있음
```

    - constraint violation 일으키는 것 없이 tuple insert 어떻게 함?
      1. person insert 하기 전에 mother, father 먼저 insert
      2. father, mother null로 초기화 → 모든 person insert 후 update

    (만약 father, moter attribute가 not null이라면 불가능한 방법)

+) complex check condition 설정 → 꼭 지켜야 하는 조건들 설정

- *assertions*
  - database가 항상 안전하길 바라는 조건을 표현함
  - example
    - student relation 내부에 있는 각 tuple에 대해 tot_cred attribute의 값이 반드시 student가 성공적으로 이수한 수업의 합과 같아야 함.
    - instructor는 semester에 같은 학기에 두 개의 다른 classroom을 가르칠 수 없음

```
create assertion <assertion-name> check (<predicate>) ⇒해당조건 충족 → 저반
//이러한 형식으로 위의 조건을 작성할 수 있음
//작성해볼 것
```

▼ SQL Data Types and Schemas

▼ Built-in Data Types in SQL

- *date* : Dates, containing a (4-digit) year, month and date
  - example: date '2005-7-27'
- *time* : time of a day → 시, 분, 초

- example: time '09:00:30' time '09:00:30.75'
- *timestamp* : date + time of day
  - example: timestamp '2005-7-27 09:00:30.75'
- *interval* : period of time
  - example: interval '1' day
  - interval value에서 date, time, timestamp subtracting
  - interval value : 다른 data type에 더해질 수 있음

▼ Large-Object Types

- Large Objects (photos, videos, CAD files, etc.)

  ⇒ Large Object로 저장됨(객체로 처리)

  - biob : binary large object → 해석되지 않은 binary data의 large collection
  - clob : character large object → character date의 large collection
- 데이터베이스 자체를 저장 x → 큰 파일, 큰 object는 다른 file에 저장

  ⇒ 그 file의 offset 저장(훨씬 더 table 작아짐)

▼ User-Defined Types

필요한 경우에 user가 type을 define할 수 있음

```
create type Dollars as numeric (12,2) final
//final : 정의가 끝났음을 알려주는 문구
```

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

- Domains : domain도 user가 define 할 수 있어야 함 (type과 유사함)

```
create domain person_name char(20) not null
//null이 아니라는 조건을 가진 char(20) 타입의 domain 'person-name'
```

```
create domain degree_level varchar(10) //degree_level : domain 설정⇒이름
    constraint degree_level_test //constraint : 제약 조건 설정
    check (value in ('Bachelors', 'Masters', 'Doctorate'));
```

- index creation
  - 내가 원하는 data를 빠르게 찾아가기 위해 index 자료구조 사용
    - 모든 tuple의 data를 확인하지 않아도 된다!

```
create index <name> on <relation-name> (attribute);
                                        //target relation, attribute
```

▼ Authorization

view → 권한이 있는 user에게만 제공

(all, none or combination)

- **privilege**라고도 불림 ⇒ relation, view와 같은 data base의 연산 특권이 있어야만 사용 가능
  - Read : reading 허용, data 수정 불가
  - Insert : new data insertion 허용, 수정 불가
  - Update : modification 허용, data 삭제 불가
  - Delete : data 삭제 허용
- database schema를 수정하는 권한(database 내부)
  - Index : creation, deletion 가능
  - Resources : 새로운 relation 생성 가능
  - Alteration : relation에 새로운 attribute 추가하거나 삭제 가능
  - Drop : relation 삭제 가능

▼ Authorization Specification in SQL

- *grant* : 권한을 구별하기 위해 사용하는 상태

```
grant <privilege list> on <relation or view> to <user list>
                          //권한 list    //target
```

(table)

- <user list>
  1. a user-id 나열
  2. *public* : 모든 사람이 사용 가능함
  3. a role : 일정 역할을 가지고 있는 사람들 group
- example

```
grant select on department to Amit, Satoshi
// 두명에게 department에 대한 selet 권한을 부여
```

- view에 대한 권한 → 해당 relation과 연관 x, src table에도 영향 x
- 권한 주는 사람도 무조건 해당 권한을 가지고 있어야 함 ☆

▼ Privileges in SQL

- *select*(read) : relation, view에 접근해서 read할 수 있는 권한

```
grant select on instructor to U1, U2, U3
//instructor table에 관한 select 권한을 user 1,2,3에게 grant
```

- *insert* : insert tuple 권한
- *update* : SQL state update 권한
- *delete* : delete tuple 권한
- *all privileges* : 모든 privileges를 가지는 short form으로 사용

▼ Revoking Authorization in SQL

- *revoke* : 권한을 빼앗아버리기..

```
revoke <privilege list> on <relation or view> from <user list>
```

(revokee - list)

```
revoke select on student from U1,U2,U3
```

- <privilege-list> = ***all*** : 가지고 있는 모든 권한 다 뺏기
- <revokee-list> = public 포함 : 특별히 권한 받은 애들 제외하고 모든 user 뺏김
- 같은 user에게 똑같은 privilege를 두 번 granted 했다면 revocation이 끝나고 privilege 유지함.
- revoked 된 권한에 따라 모든 권한 또한 revoked 될 수 있음

- 권한에 대한 특별한 조건 ? → 딱히 없음. 해당 권한이 있는 사람만 뺏어야 할 것

▼ roles

database에 access/update 할 수 있는 user들을 다른 user와 구분

- to create

```
create a role <name>

//example
create a role instructor;
```

- role이 create 되고 나서 user에게 해당 role 부여할 때

```
grant <role> to <users>

//example
 grant instructor to Amit;
```

- example

```
grant select on takes to instructor;
//role, privilege 설정 한 번에 가능
```

```
create role teaching_assistant
grant teaching_assistant to instructor;
//instructor가 teaching_assistant의 모든 권한 다 가능해야 함 -> role 물려줌
```

```
//chain of roles
create role dean;
grant instructor to dean;
grant dean to Satoshi;
```

▼ Autorization on Views

```
//geo_instructor view 제작
create view geo_instructor as
    (select *
    from instructor
    where dept_name = 'Geology');
grant select on geo_instructor to geo_staff
//geo_instructor에 대한 권한을 geo_staff에게 줌
```

```
//만약에 geo_instructor에 대한 모든 tuple select 요청
select *
from geo_instructor;
```

1. geo_staff : instructor에 대한 권한 없다면
   -> 상관 없음 view에 대한 권한 굳이 안 가져도 됨 → table에 대해서는 굳이 X
2. creater of view : instructor에 대한 권한을 안 가져도 되는가?
   -> 가져야 됨 ! view 생성.