Chapter 2. Intro to Relational Model

relationship - 데이블의 각 행은 일련의 값 사이의 relationship 을 표현
↳ relation = tuple 의 집합
↳ 하나의 tuple = 하나의 행
↳ 하나의 attribute = 하나의 속성

relation = domain → 각 속성은 domain 이라 하는 허가된 값의 집합을 가짐 (→ 그냥 type ex. String, Int)
ex) name 속성의 domain은 가능한 교수 이름
→ atomic 이어야함 ex) 전화번호! → atomic
나눌수 X 지역번호 + 국가번호 → atomic X

- data schema vs database instance
database의 논리적 설계 데이터의 스냅샷

- key
tuple 의 속성값은 그 tuple을 유일하게 식별할 수 있어야 함.
→ 두 개의 tuple 의 모든 속성값이 같으면 안됨.

super key : 한 relation 에서 그 tuple 을 유일하게 식별할 수 있도록 해주는 하나/그 이상의 속성 집합
ex) ID는 super key O, name은 super key X, but (ID, name) 은 super key O
K가 super key 라면 K포함 어떤 집합도 super key
super key 의 부분집합이 super key 이될 수 있음
→ 최소한의 슈퍼 키 = candidate key (후보키) → superkey가 너무 많아질 수 있으니 최소한만! ⇒ candidate key
ex) ID 후보키 O, super key O
(name, dept_name) 후보키 O
(ID, name) 후보키 X, super key O
⇒ 선택된 candidate key = primary key

- primary key constraint : 절대로 변하지 않거나 매우 드물게 변하도록 선택
- foreign-key constraint : r1 에 존재하는 튜플의 A 값이 r2에 있는 어떤 튜플의 B 값으로 나와야 한다
V 관계
referencing relation — foreign key — referenced relation
- referential integrity constraint : 참조하는 relation 의 어떤 tuple 의 특정 속성이 참조된 relation 의 한 튜플의 특정 속성으로 존재

query langage
- imperative query langage : 연산 수행 명령
- functional query langage : 함수 실행으로 표현, 갱신 X
- declarative query langage : 정보만 기술, 단계/순서 X → pure query.

selection : σ ex) $\sigma_{salary > 90000}(instructor)$

project : Π ex) $\Pi_{ID, name, salary/12}(instructor)$
(12로 나눠서 보여줌)

selection × project ex) $\Pi_{name}(\sigma_{dep\_name = "Physics"}(instructor))$
→ 물리학과에 속한 모든 교수들을 찾아라!

cartesian product : × 그냥 단순데카르트 곱이라고

→ 같은 속성은 instructor.ID, teaches.ID 로 구별

Join : σ & × ex) $\sigma_{instructor.ID = teaches.ID}(instructor \times teaches)$

$= instructor \bowtie_{instructor.ID = teaches.ID} teaches$

Union : ∪ (→ 합수)
- r라는 R과 s가 같은 두의 속성을 가질 때
- r의 i번째 속성의 domain 과 s의 i번째 속성 domain이 같을 때

차집합 : − 같은 것은.

배정 : ← ex 임시 릴레이션 ← 조건.

재명명 (rename) : 관계 대수의 결과에 이름을 주는 데 유용. ρ

ex) $\sigma_{i.salary > w.salary}(\rho_i(instructor) \times \sigma_{w.id = 12121}(\rho_w(instructor)))$

연습문제

2.1 person_name

person_name

company_name

2.14 - d

$\Pi_{ID, person\_name}$ (employee $\bowtie_{employee.ID = work.ID}$ work) $\bowtie$ (company.name = work.name $\cap$ company.city = employee.city (company))

2.15 - b

Π_{ID} (customer $\bowtie$ ($\sigma_{balance > 6000}$ (depositor $\bowtie$ account))

2.17 - c

$\Pi_{customer\_name}$ (customer $\bowtie_{balance > 6000 \cap branch\_name = \text{"Uptown"}}$ (depositor $\bowtie$ account))

2.18 - b

$\Pi_{ID, name}$ (instructor $\bowtie$ (department.dept_name = instructor_dept_name ($\sigma_{building = \text{"watson"}}$ department))

2.18 - c

$\Pi_{student.ID, student.name}$ ($\sigma_{dept\_name = \text{"comp.sci"}}$ (student $\bowtie_{student.ID = takes.ID}$ (takes $\bowtie_{takes.course\_id = course\_id.course.id}$ course)))

2.18 - d

$\Pi_{ID, name}$ (student $\bowtie$ (student.ID = takes.ID ($\sigma_{year = 2018}$ takes)))

student. student.

2.18 - e

$\Pi_{student.ID, student.name}$ (student)

$-$ $\Pi_{student.ID, student.name}$ ($\sigma_{year = 2018}$ (student $\bowtie_{student.ID = takes.ID}$ takes))

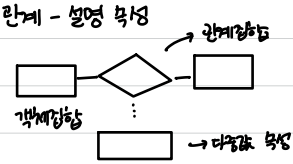
Chapter 6.

위험성 ┌ 중복성
└ 불완전성 → 주 키에 null 저장

개체집합 ≒ 속성 집합

개체가 행하는 기능 = 역할 (role)

관계 - 설명 속성



E-R 모델 ┌ 단순 속성 simple attribute
└ 복합 속성 cosposite attribute

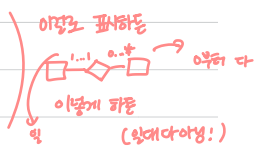
┌ single-valued attributed
└ multivalued attributed

- derived attributed

mapping cardinality / cardinality ratio

바이너리 관계에서는 화살표 최대 하나!

┌ 일대일 ⟷
├ 일대다 ⟵
├ 다대일 ⟶
└ 다대다 ——



┌ 전체참여 → 이중선 적어도 하나의 관계에 참여해야 하는 경우
└ 부분참여 일부 개체가 참여하지 않을 수도 있는 경우

관계 집합 → primary key(E1) ∪ primary key(E2) ··· ∪ {a1, a2, a3 ... } ← 추가속성

일대다 / 다대일 경우) "다" 쪽의 주 키가 슈퍼키 & primary key

일대일 인 경우) 둘 중 하나를 primary key 로 선택 가능

바이너리 관계인 경우) cardinality 고려 X 면 합집합으로 만든 superkey 가 유일한 candidate & primary key

화살표는 최대 하나!

- 약한 개체 집합

중복제거방법 ① sec_course 자체 제거

② section에 course_id 저장 안하고 나머지만 저장

→ 하지만 식별하기 위해선 course_id 필요!!

→ 약한 개체 집합 ——의존——> 식별 개체 집합

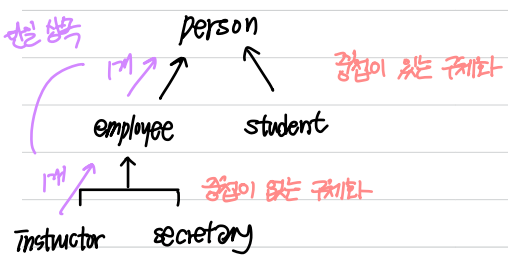
→ ∴ 해의 후키

identifying entity set의 primary key

+ weak entity set의 primary key

- 다중키도 후키 가능!
- 만약 개체 집합이 primary key, multivalued attributed 만 지녔다면,

그 개체 집합에 대한 relation schema는 primary key 만 지닐 것이다.



chapter 3

DDL - relation schema 정의

DML - tuple 정의

Integrity

⋮

char

varchar 이거써.

int

smallint

numeric

real

double precision

float

null 값가능

drop table r 전부삭제

delete table r 부분삭제

create table r 생성

alter table r add A D r에 A에 D형 생성

alter table r drop A 〃 삭제

select — 중복제거 : distinct / 중복허용(생략이) : all

from — 이름 변경 : A as B A를 B로 변경

where — and / or / not 사용가능

% 문자는 어떤 유효 문자열과 일치

_ 문자는 어떠한 문자와 일치

\ escape 문자

→ ex) _ _ _ 세 문자로 이뤄짐 — 일치

_ _ _ % 세 문자 이상으로 이뤄짐 — 일치

Intro % Intro로 시작하는 —

% Comp % Comp 가 속하는 —

order by (desc, asc 명시가능)

ex) order by salary desc, name asc

이것부터 먼저 → 같으면 그다음 이거

∪ union
∩ intersect
− except

) 중복 자동 제거 (중복 허용하고 싶으면 union all)

null + 5 = null 이어야 함

∴ 이러한 null을 포함하는 결과를 unknown으로 처리

↳ true ∧ unknown → unknown
false ∧ unknown → false
unknown ∧ unknown → unknown

true ∨ unknown → true
false ∨ unknown → unknown
unknown ∪ unknown → unknown

∼ unknown = unknown

avg
min
max
sum
count (count * 로 작성 / distinct 사용 금지!)

) null 무시

count → null = 0 으로 간주

group by - 단일 집합이 아니라 복수의 튜플 집합에 대하여 집계함수 적용하고 싶을 때!

ex) select dept_name, avg salary
from instructor
group by dept_name

→ 이것으로 group 화 함!

→ group by의 각각 group에 적용되는 조건.

∴ group by 에 없는 속성이 select 에 나오면 안됨
나오려면 집계 함수써서 나오기!

having → group by의 각각 group에 적용되는 조건.

in → 집합 멤버십을 test.

→ 중복 제거함 ∴ distinct 를 사용해야 함!

not in 도 가능.

집합비교 — > some → 하나 이상보다 큰

< some , <= some, <> some ···

< all → 전부는 아닌

<=all, >=all, <>all ···

< > some ≠ not in

= some = in

< > all = not in

= all ≠ in

빈 relation — exists → true / false 반환

A > B → not exists (B except A)

unique — 최대 1번! check → 빈집합도 true

not unique - 최소 2번!

with 절에서만 유효한 임시 relation

scalar 하위 질의가 하나의 속성을 가지는 오직 하나의 tuple 만 반환한다면, 값 반환되는 어떤 곳에서도 나타날 수 있게 함.

3.3

```
update instructor
  set salary = salary x1.1
where dept_name = "Comp.Sci"

delete frome course
where course_id not in ( select course_id
                         from section )

insert into instructor
       select ID, name, dept_name, 10000
       from student
       where tot_cred > 100
```

3.4

```
select count distinct driver_id
from person, accident, owns, participated
where accident.report_num = part.report_num
   and owns.driver_id = part.driver_id
   and owns.license_plate = part.license_plate
   and year = 2017

delect from care
 where license_plate in (select lincese_plate
                         from owns o
                         where o.driver_id = '12345')
```

3.5

```
select ID
    case when score<40 —
         :
         else
    end
from marks
```

```
with grades as
(select ID
    case when score<40 —
         :
         else
    end as grade
from marks)
select grade, count(ID)
from grades
group by grade
```

```
with grades as
(select ID
    case when score<40 —
         :
         else
    end grade
from marks)
select grade, count(ID)
from grades
group by grade
```

3.6

```
lower(—) like "%sci%"
```

3.8

```
select ID
from customer, account, depositor
where customer.ID = depositor.ID
      and account.num = depositor.num
except
select ID
from customer, loan, borrower
where customer.ID = borrower.ID
      and loan.num = borrower.num
```

$\Pi.\Pi \Rightarrow$

```
(select ID
from depositor)
except
(select ID
from borrower)
```

```
select ID
from customer as F, customer as S
where F.street = S.street
      and F.city = S.city
      and S.name = '12345'
```

```
select distinct branch-name
from branch, customer, depositor, account
where c.ID = d.ID
    and c.city = " Harrison
    and account.num = depositor=num
```

3.9

```
select ID, city, name
from employee, works
where employee.ID = works.ID
    and company_name = " First — "
```

```
select ID, person_name, city
from employee, works
where employee.ID = works.ID
    and company_name = " First — "
    and salary > 10000
```

```
select ID                    company_name <> First —
from works
where company_name not in (select company_name
                           from works
                           where company_name = " First — ")
```

ID ID

```
select ID
from works
where salary > all
    (select salary
     from works
     where company_name = " ")
```

```
select company_name
where company
where city > some ( select city
      not exists      from   company
                      where  company_name = " — "
                      except — where. S.compan- T.company_name
select company_name , count (ID) as A
from works
group by company_name
having count (distinct ID) >= all
        ( select count distinct ID
          from works
          group by company_name )
```

```
select company_name
from works
group by compan_name
having salary > (select avg salary
     ^           from works
    avg          where  —  =  — )
```

3.10

```
update employee
  set city = 'Newtown'
  where ID = '12345'

update works
  set salary = salary *
        (case
            when salary < 100000 then 1.1
            else 1.03
        end )
```

```
where  ID (select ID
          from  manages
          where manages.ID = works.ID)
       and company_name = —
```

4.2

```
select  ID, count ( section_id) as number_taught
from  Instructor natural left outer join teaches
group by ID
```

```
select  ID,
    (select count (*) as num —
     from teaches T
     wher  T.id = I.id)
from Instructor I
```

```
select  ID, name
    decode (name, null, '—', name) as name
from (section  natural left outer join teaches ) natural left outer
                                                  join Instructor
where  year = 2018
   and semester = spring
```

```
select count (ID), depart_name
from  department  natural left outer join Instructor
group by depart_name
```

4.3

```
select *
    from student natural join takes
    union
    select *
    from student
```

4.15

```
select *
from section join classroom using (building, room_number);
```

(join: inner)

4.17

```
select ID
from student left outer join advisor
where advisor.id is null
```

2.18

a. $\Pi_{ID, name}(\Delta_{department\_name = \text{"Painter"}}(instructor))$

b. $\Pi_{ID, name}(instructor \bowtie_{instructor.dept = dept\_dept} (\Delta_{building = \text{"Watson"}}(department)))$

c. $\Pi_{ID, name}(\Delta_{dept = Cou.}(student \bowtie_{stu.id = takes.id} takes \bowtie_{takes.id = course} \ldots))$

d. $\Pi_{ID, name}(\Delta_{year = 2018}(student \bowtie_{stu.id = take.id} takes))$

e. $\Pi_{ID, name}(student) - \Pi_{ID, name}(student \bowtie_{ct} \ldots)$

2.14

a. $\Pi_{name}(\Delta_{company = bigbank}(employ \bowtie_{m} \ldots (works))$

b. $\Pi_{ID, name, city}(employ \bowtie (person.name = (\Delta_{n} \ldots$

c. $\Pi_{ID, na} - \ldots (employ \bowtie$

$\cap$

3.11

a. select distinct ID, name
from student, takes, course.
where student.id = takes.id
and course.dept = "Comp.Sci"

b. select distinct s.ID, s.name
from student . s
where not exists (
select *
from takes
where 2017 > year
and s.id = t.id

c. select max salary., dept_name
from instructor
group by depart_name

d. with max_sal (dept_name) as (
select max salary., dept_name
from instructor
group by depart_name
)
select min(max_sal)
from max_sal

3.12

a. insert into course (_ _ _ _)
values (_ _ _ _)

b. insert into

c. insert into takes (
select ID
from student
where comp. = 'c'
)

d. delet from takes
where .id = 12345)
and

f. delect from takes
wher course.id in (
select course.id
from course
where title = '%. %'
cover like

4.2 a. select ID, count (sec_id) as SE
from section natural left outer join teaches
group by ID

b. c. select ins.id ins.name
decode (name, '-', null, name) as ins.name
from ~~instructor~~ left ~~—~~ ~~section~~
section teaches.

ID

```
select distinct course.id ~~student.ID~~
from takes
group by ID, course.id
having count * > 2
order by course.id
```

```
select ID
from works
where a.salary > all
        (select b.salary
         from works
         where name = "small")
```

```
select ID
from student left outer join takes using course.id
where course.id = null
```

24) each student retacken course at least twice
course ID, student's ID

```
select distinct course.ID, ID
from course, student
where not unique (select cours.ID, ID
                  from course, student, takes
                  where course.cours Id = takes. course.Id
                      and student.ID = takes.ID)
```

⇓

```
select distinct course.ID, ID
from takes
 group by ID, course.ID
 having count * > 2
 order by course_ID
```

```
select ID, person_name, city
from employee, works
where employee_.ID = works.ID
      and works.company_name = "First Bank Corp"
```

```
select ID
from works ~~f, works, s~~
where ~~s~~.salary > ~~max~~ all (select salary
                                 from ~~f~~ works
                                 where ~~f~~ company_name = "Small Bank Corp")
```