4,5,6. Basic SQL

▼ Basic Query Operation

▼ Rename Operation

realtion, attribute 이름 변경 가능

```
select ID, name, salary/12 as monthly_salary
//salary/12 -> monthly_salary로 attribtue 이름 변경(연산의 경우 데이터가 숫자형이어야 함
from instructor
```

이름 간단하게 치환 가능 → 같은 table인데 서로 다른 table로 인식해서 연산하고 싶을 때

```
instructor as T ≡ instructor T
```

▼ String Operation

SQL : char에 대해 비교 연산자 제공

1. *percent*(%) : 해당 string을 포함하는 모든 string
    a. 문자로서 % 포함하고 싶다면 백슬래쉬(\) 사용
2. *underscore*(_) : 작성된 개수만큼 들어감
3. *like* : 특정 패턴을 가진 char

```
Intro% //Intro로 시작하는 string
%Comp% //중간에 Comp가 들어가는 string
_ _ _ //3글자 string
_ _ _ % //최소 3글자 string
% _ _ _
//1
select name
from instructor
where name like '%dar%'
//dar을 포함한 string 모두 고름
```

4. *concatenation*(||) : 두 string을 이어 붙힘
5. *UPPER(), LOWER()*
6. *LENGTH(), SUBSTRING(str, position, length)*

▼ Ordering the dispaly of Tuples

데이터는 기본적으로 정렬이 되지 않은 상태 → 기준이 되는 attribute 제공해야 함.

order by : 알파벳 순서로 정렬 → **asc**/**desc**(오름(default)/내림)

```
select distinct name
from instructor
```

```
order by name //기준을 여러 개 설정할 수 있음
```

↳ 하나가 아니고 여러개일수도.

▼ Where문

- *between* comparison operator
  - *select* 뒤에 사용 → 어떤 기준으로 정확하게 attribute 명시하고 있는지 봄

```
//연봉이 9만불-10만불 사이인 모든 교수의 이름을 찾아라
select name
from instructor
where salary between 90000 and 100000

//tuple comparison → 하나의 tuple 내부에서 비교 ⇒ Biology 학과에서 가르친 교수의 이름과 스윙 전 모두조회
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology')
//두개의 조건을 모두 충족하는 query만 출력
```

비교 / 비교

↳ 하나의 tuple 내에서 비교

↳ 충족하는 모든 tuple select.

교수테이블에서만 있는 attribute.

▼ Set Operations

union, *intersect,* and *except*

- all duplicate : union all, intersect all, except all

```
//2017년 가을 혹은 2018년 봄에 열린 강의
(select course_id from section where sem = 'Fall' and year = 2017)
union ⇒OR
(select course_id from section where sem = 'Spring' and year = 2018)

//2017년 가을과 2018년 봄 모두 열린 강의
(select course_id from section where sem = 'Fall' and year = 2017)
intersect ⇒and
(select course_id from section where sem = 'Spring' and year = 2018)

//2017년 가을에는 열렸지만 2018년 봄에는 열리지 않은 수업
(select course_id from section where sem = 'Fall' and year = 2017)
except ⇒차집합 (but not in)
(select course_id from section where sem = 'Spring' and year = 2018)
```

▼ Null Values → unknown / does not exist

특정 attribute에 대해 null value 가질 수 있음

- *null* : an unknown value 혹은 존재하지 않는 value를 나타냄
- *null*을 포함한 산술 연산의 결과는 null임 ex. 5+null = null

```
//연봉이 null인 모든 교수의 이름
select name
from instructor
where salary is null
```

↳ null value check하는데 사용됨.

- null logic
  1. null이 들어가 있는 모든 비교 연산자 → unknown
  2. 진리값

OR: (unknown or true) = true,
(unknown or false) = unknown
(unknown or unknown) = unknown

AND: (true and unknown) = unknown,
(false and unknown) = false,
(unknown and unknown) = unknown

NOT: (not unknown) = unknown

"P is unknown" evaluates to true if predicate P evaluates to unknown

  3. 만약 unknown 값이 있다면 where문의 결과가 false로 간주됨

▼ Aggregate Functions

operate on the multiset of values of a column of a relation, and return a value

- *avg, min, max, sum, count*(number of values)

```
//컴공 교수들의 평균 연봉
select avg (salary)
from instructor
where dept_name= 'Comp. Sci.'; //조건 설정 -> 조건에 맞는 attribute만 계산

//2018년 봄학기에 수업을 한 교수의 총 인원 수
select count (distinct ID) //distinct : 중복 제거
from teaches
where semester = 'Spring' and year = 2018;

//coure relation의 tuple의 개수
select count (*)
from course;
```

- *group by*

```
//Group By 사용 -> 각 학과의 교수님들의 평균 연봉
//해당되는 attribute만 작성해야 함
select dept_name, avg (salary) //select에 ID 작성 -> 제대로 된 쿼리가 아님
from instructor
group by dept_name;
```

select

departments with no instructor will not appear in result

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Instructor

  - select문의 attribute는 모두 반드시 group by에도 존재해야 함
  - group이 생성된 후 where문 사용 → 잘못된 결과가 나올 수 있음. 에러는 X

- *having* clause : group 이후에 적용 ↔ where : group이 만들어지기 전에 사용

```
//평균 연봉이 42000 넘는 모든 학과의 평균 연봉과 이름 찾기
select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name
having avg (salary) > 42000; //group에 대한 조건 설정
//where avg(salary) > 42000 => group이 만들어지기 전에 where 사용해야함.
```

- *null* value (unknow value)

```
//총 연봉 구하기
select sum (salary )
from instructor
```

  - 만약 우리가 모르는 값에 더하면 null value 제외하고 생각함
  - *null* value가 아니면 return null
  - count를 제외한 모든 aggregate operation = aggregated attribute의 *null* value 무시

▼ Nested Subqueries

select 안에 다른 select가 존재

- subquery : 다른 query 안에 들어가 있는 select-from-where 문을 뜻함 ⇒ query 내부 특정 부분에 존재

  → 집합 대 집합을 비교 혹은 내가 확인하고 싶은 값이 포함되었는지 확인

Course_id	Sec_id	Semester	Year
BIO-101	1	Summer	2017
BIO-301	1	Summer	2018
CS-101	1	Fall	2017
CS-101	1	Spring	2018
CS-190	1	Spring	2017
CS-190	2	Spring	2017
CS-315	1	Spring	2018
CS-319	1	Spring	2018
CS-319	2	Spring	2018
CS-347	1	Fall	2017
EE-181	1	Spring	2017
FIN-201	1	Spring	2018
HIS-351	1	Spring	2018
MU-199	1	Spring	2018
PHY-101	1	Fall	2017

봄에

```
//2017년 가을과 2018년 모두 열린 강의
select distinct course_id   ⇒ CS101
from section
where semester = 'Fall' and year= 2017 and   → CS101, CS347, PHY101
    course_id in
(select course_id from section
    where semester = 'Spring' and year= 2018);   → CS101, CS315, CS319, FIN201, HIS351, MU199
```

비교해서 공통인 값 ⇒ "CS101"

```
//2017년 가을에 열렸지만 2018년 봄에는 열리지 않은 강의   ⇒ 포함되지 않는 value만
select distinct course_id                                  → CS347, PHY101
from section
where semester = 'Fall' and year= 2017 and
    course_id not in
    (select course_id from section
        where semester = 'Spring' and year= 2018);
```

```sql
//ID 10101인 교수님 수업을 들은 학생 총 수
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
    (select course_id, sec_id, semester, year
        from teaches
        where teaches.ID= 10101);
        //내부 쿼리로 해당 attribute에 대한 table 생성 -> 그 중 id 총 개수 세기
        //최선의 방법은 아님. 또 다른 방법으로 작성 가능
```

▼ Set Comparison

```sql
//컴공 교수 중 단 한 명이라도 연봉이 더 많은 사람
//두 명 다 보다 많으면 두 번출력 -> distinct로 중복 제외
//1.
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'CSE';

//2.
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'CSE');
```

```
sqlite> select * from instructor;
22222|Wookhee|CSE|600000
11111|John|EE|5000000
33333|Jane|IE|5000000
12345|Kane|CSE|5000000
23456|Smith|EE|6000000
34567|Kate|IE|6000000
```

- F <comp> **some** r : $\exists t \in r$ such that (F <comp> t)
  - comp : 비교 연산자 $<, \leq, >, =, \neq$

(5 < some {0, 5, 6}) = true (read: 5 < some tuple in the relation)

(5 < some {0, 5}) = false

(5 = some {0, 5}) = true

(5 ≠ some {0, 5}) = true (since 0 ≠ 5)

(= some) ≡ in
However, (≠ some) ≠ not in

```sql
//생명과의 모든 교수의 연봉보다 높은 교수의 이름
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

- F <comp> **all** r : $\forall t \in r$ (F <comp> t)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

(5 < all {0, 5, 6}) = false 모두가 5보다 큰 건 아님

(5 < all {6, 10}) = true

(5 = all {4, 5}) = false

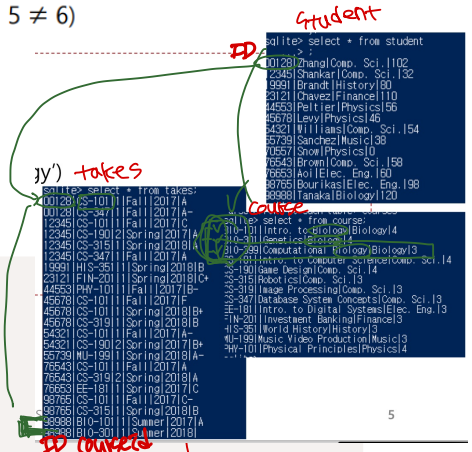
(5 ≠ all {4, 6}) = true (since 5 ≠ 4 and 5 ≠ 6)

attribute 형태는 같아야 함.

▼ Empty Relations

▼ exist : Ø가 아니면면 true를 return

- ***exists*** r : r ≠ Ø
- ***not exists*** r : r = Ø



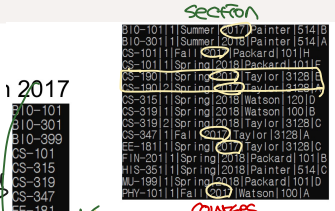

```
//biology 학과에 개설된 모든 수업을 들은 학생 다 찾기
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                  except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

- Correlation Variables ☆ ⇒ Correlated subquery.

```
//2017년 가을 학기와 2018년 봄 학기 모두 열린 강의를 다 찾기
select course_id
from section as S
where semester = 'Fall' and year= 2017 and
    //서브 쿼리를 이용하여 두 조건 모두 존재하는 tuple 찾기
    exists (select *
            from section as T
            where semester = 'Spring' and year= 2018 and S.course_id= T.course_id)
```

2018년 봄 학기 열린 강의 ∩ 2017년 가을학기에도 열린 강 찾기

→ Correlated subquery



▼ Duplicate Tuples

▼ unique : 한 번도 열리지 않은 강은 나오지 X

subquery에 중복된 tuple이 존재하는지 아닌지 test ⇒ 중복 tuple 존재 X

- empty set : true


```
//2017년에 한 번 이상 열린 강의
select T.course_id
from course as T
where unique (select R.course_id //같은 녀석이 아예 없을 때에도 결과가 나옴 → true
    from section as R
```

↔ distinct : 없는 tuple은 false

```
        where T.course_id= R.course_id
        and R.year = 2017);
```

▼ From

from 문을 사용해서 subquery

```
//평균 연봉이 300달러 이상인 사람들의 평균 연봉
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 3000000;
```

```
//subquery의 relation 이름을 rename
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
        from instructor group by dept_name)
        as dept_avg (dept_name, avg_salary) where avg_salary > 3000000;
```

▼ With ⇒ logical level 변경X

일시적인 view를 제공하는 query (사용자에게 특화된 임시적인 table)

```
//maximum budget을 가진 모든 학과
//with -> logical level을 변경하지는 않음.
with max_budget (value) as
        (select max(budget)
        from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

- 복잡한 query을 작성하는데 유용
- 대부분의 database system(minor syntax variation)에서 쓰임

```
//총 연봉이 모든 학과의 총 연봉의 평균보다 높은 학과 찾기
with dept_total (dept_name, value) as
        (select dept_name, sum(salary)
        from instructor
        group by dept_name),
        //interpreter -> sequential하게 작동, 순서대로 정의해주고 작성해야 함
        dept_total_avg(value) as
        (select avg(value)
        from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

```
sqlite> with dept_total (dept_name, value) as
   ...>             (select dept_name, sum(salary)
   ...>              from instructor
   ...>              group by dept_name),
   ...>         dept_total_avg(value) as
   ...>             (select avg(value)
   ...>              from dept_total)
   ...> select * from dept_total_avg;
9200000.0
sqlite> with dept_total (dept_name, value) as
   ...>             (select dept_name, sum(salary)
   ...>              from instructor
   ...>              group by dept_name),
   ...>         dept_total_avg(value) as
   ...>             (select avg(value)
   ...>              from dept_total)
   ...> select * from dept_total;
CSE|5600000
EE|11000000
IE|11000000
```

▼ Scalar Subquery

single value 반환하는 query → row가 하나만 나와야 함(tuple 하나)

- tuple이 하나 이상 나오면 runtime error 발생

```
//select문 -> 해당 학과의 교수진 인원 수
select dept_name,
    (select count(*)
        from instructor
        where department.dept_name = instructor.dept_name)
    as num_instructors
from department;

//where문 -> 해당 학과의 교수 중 budget이 연봉의 10배보다 작은 교수
select name
    from instructor
        where salary * 10 > (select budget from department
                                where department.dept_name = instructor.dept_name)
```

▼ Modification of the Database

- ***Deletion***

```
//delete all instructors
delete from instructor

//delete all instructors from the Finance department
delete from instructor
where dept_name = 'Finance';

//delete 왓슨 건물에 위치한 학과와 관련된 모든 교수들의 모든 tuple
delete from instructor
    where dept_name in (select dept_name
                        from department
                        where building = 'Watson');

//delete instructor의 평균 연봉보다 적은 연봉을 가진 모든 교수 delete
delete from instructor
where salary < (select avg (salary) from instructor);
//problem : tuple 지울 때 avg(salary) 값이 변경됨
//solution in SQL
//1. avg 계산 후 모든 tuple 찾아서 지우기
//2. avg 계산 없이 모든 tuple 지우기
```

- ***Insertion***

```
//course table에 새로운 tuple add
insert into course values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
insert into course (course_id, title, dept_name, credits)
                                    values ('CS-437', 'Database Systems', 'Comp

//tot_creds이 0으로 설정되어 있는 모든 교수를 student relation에 add
insert into student
    select ID, name, dept_name, 0
    from instructor
```

- insertion 된 결과가 완전히 다 insertion 하기 전에 검사함
  - insert into table1 select * from table1 ⇒ problem 일으킴

- ***Updating***

```
//연봉이 10만불 넘는 교수들의 연봉을 3퍼 인상, 나머지는 5퍼 인상
update instructor
    set salary = salary * 1.03
    where salary > 100000;

update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

반전...

```
//same example -> better
update instructor
    set salary = case
        when salary <= 100000 then salary * 1.05
        else salary * 1.03
        end
```