# 데이터베이스 시스템 3장

| 태그 | 데이터베이스 시스템 |
| --- | --- |

# 3.1 SQL 질의어 개요

- SQL (Structured Query Language)
- DDL(Data-definition language), DML(Data-manipulation language), Integrity, View definition, Transaction control, Embedded SQL, Dynamic SQL, Authorization으로 구성
- 표준이 있지만 벤더사마다 지원할 수도 안할 수도 있음.

# 3.2 SQL 데이터 정의

DDL로 정의할 수 있는 것들

- 각 릴레이션의 스키마
- 각 속성의 타입
- 무결성 제약 조건
- 각 릴레이션을 위해 만들어진 인덱스 집합
- 각 릴레이션에 대한 보안 및 권한 부여 정보
- 디스크에 저장된 릴레이션의 물리적 저장구조

## 3.2.1 기본 타입

모든 타입은 null값 포함 가능

- char(n)
  - 고정 길이 문자
  - 크기가 n보다 작으면 공백을 채워서 매꿈, 문자열 비교시에도 동일
  - char과 varchar를 비교시 위의 예시처럼 문자가 채워질 수도 있고 안 채워질 수도 있어서 **varchar 사용 추천**
- varchar(n)
  - 가변 길이 문자

- int
  - 정수
- smallint
  - 작은 정수
- numeric(p, d)
  - 사용자가 지정한 정밀도를 가짐
  - p개의 숫자 중 d개는 소수점 이하에 있는 숫자의 개수
- real
  - 시스템에 따라 달라질 수 있는 정밀도를 가지는 부동 소수점 수
- double precision
  - 시스템에 따라 달라질 수 있는 정밀도를 가지는 배정밀도 부동 소수점 수
- float(n)
  - 적어도 n개의 숫자를 나타낼 수 있는 정밀도를 갖는 부동 소수점 수

## 3.2.2 기본 스키마 정의

```
create table department (
    dept_name varchar(20),
    building varchar(15),
    budget numeric(12, 2),
    primary key (dept_name)
);
```

- primary key
  - 릴레이션의 주키를 정의
  - 주키는 null이 아니거나 유일해야 함
  - 생략 가능하나 명시하는 것을 권장
- foreign key
  - 어떤 튜플의 속성 값이 관계가 있는 릴레이션의 일부 튜플의 주 키 속성값에 상응해야 함
  - foreign key (외래 키 속성) references 참조되는 테이블(참조되는 속성)

- not null
  - 도메인에 대한 제약 조건
- DBMS는 무결성 제약 조건을 위반하는 모든 갱신을 막음
- drop table r;
  - 스키마까지 삭제
- delete from r;
  - 스키마는 삭제하지 않음
- alter table r add A D;
  - A는 속성의 이름, D는 그 속성의 도메인
- alter table r drop A;

# 3.3 SQL 질의의 기본 구조

- select, from where →기본적인 구조

## 3.3.1 단일 릴레이션에 대한 질의

```
selet name
from instructor;

select distinct dept_name
from instructor;

select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 70000;
```

- 관계형 모델의 공식적이며 수학적 정의에서는 릴레이션은 집합
  - 중복된 튜플은 존재 X
  - SQL은 표현의 결과나 릴레이션에서도 중복 허용
- where절에 논리 접속사, 비교 연산자 사용 가능

## 3.3.2 복수의 릴레이션에 관한 질의

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name = department.dept_name;
```

- from 절에 나열된 릴레이션의 카티션 곱
- where 절에 명시된 술어 적용
- select 절에 명시된 속성 출력
- 하지마 DBMS는 이런식으로 SQL질의를 수행하지 않음
    - 내부적으로 최적화 수행

# 3.4 부가적인 기본 연산

## 3.4.1 재명명 연산

```
select name as instructor_name, course_id
from instructor, teaches
where instructor.ID = teaches.ID;

select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

## 3.4.2 문자열 연산

- SQL은 문자열을 작은따옴표를 사용하여 표기
- 대소문자 구분
    - DBMS 벤더마다 다를 수도?
- upper(s), lower(s), trim(s)
- like를 통한 패턴 매치
    - 퍼센트(%): 어떤 부분 문자열과도 일치
    - 밑줄(_): 문자는 어떠한 문자와도 일치

```
select dept_name
from department
where building like '%Watson%';
```

### 3.4.3 Select 절의 속성 지정

```
select instructor.*
from instructor, teaches
where instructor.ID = teaches.ID;
```

- 별표(*): 모든 속성

### 3.4.4 튜플 출력의 순서

```
select name
from instructor
where dept_name = 'Physics'
order by name;
```

- 기본적으로, 오름차순 항목 정렬
- asc: 오름차순, desc: 내림차순

### 3.4.5 Where 절의 술어

```
select name
from instructor
where salary between 90000 and 100000;
```

## 3.5 집합 연산

### 3.5.1 합집합 연산

```
(select course_id
from section
```

```
where semester = 'Fall' and year = 2017)
union
(select course_id
from section
where semester = 'Spring' and year = 2018);

(select course_id
from section
where semester = 'Fall' and year = 2017)
union all
(select course_id
from section
where semester = 'Spring' and year = 2018);
```

- union 은 중복 제거됨
- union all 은 중복 제거되지 않음
    - 두 집합에 포함된 모든 결과 포함

## 3.5.2 교집합 연산

```
(select course_id
from section
where semester = 'Fall' and year = 2017)
intersect
(select course_id
from section
where semester = 'Spring' and year = 2018);

(select course_id
from section
where semester = 'Fall' and year = 2017)
intersect all
(select course_id
from section
where semester = 'Spring' and year = 2018);
```

- intersect 는 중복 제거됨

- intersect all 은 중복 제거되지 않음
    - 두 집합에서 나타나는 중복 튜플 개수의 최솟값과 같음

### 3.5.3 차집합 연산

```
(select course_id
from section
where semester = 'Fall' and year = 2017)
except
(select course_id
from section
where semester = 'Spring' and year = 2018);

(select course_id
from section
where semester = 'Fall' and year = 2017)
except all
(select course_id
from section
where semester = 'Spring' and year = 2018);
```

- except 는 중복 제거됨
- except all 은 중복 제거되지 않음
    - c1의 중복된 개수에서 c2의 중복된 개수를 뺀 것에서 그 차가 양수일 때와 같음

## 3.6 널 값

- 산술 연산식의 결과는 입력 값이 널이면 널임
- 널 값을 포함하는 비교의 결과는 unknown
    - not unknown = unknown
    - true or unknown = true
    - false or unknown = unknown
    - unknown or unknown = unknown
    - true and unknown = unknown

    - false and unknown = unknown
    - unknown and unknown = unknown
- is null, is not null, is unknown, is not unknown
- distinct 시 null 값 중 하나만 선택됨

# 3.7 집계 함수

- 값의 모임을 가지며, 결과 값으로 단일 값 반환 하는 함수
- 표준 내장 집계 함수

## 3.7.1 기본 집계

- avg, min, max, sum, count
- 평균은 중복 제거 시 위험할 수도
- 집계 전 중복 제거가 필요할 수도
- count(*)는 distinct 금지, max, min에서는 가능

## 3.7.2 그룹 단위 집계

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

- group by나 집계 함수에 사용되지 않는 속성은 select에 올 수 없음

## 3.7.3 Having 절

- group by로 그룹화 된 후 적용
- Having 절에 집계 함수 사용 가능
    - group by나 집계 함수에 사용되지 않은 속성은 올 수 없음

```
select course_id, semester, year, sec_id, avg(tot_cred)
from student, takes
where student.ID = takes.ID and year = 2017
```

```
group by course_id, semester, year, sec_id
having count(ID) >= 2;
```

### 3.7.4 널 값과 불리언 값의 집계

```
select sum(salary)
from instructor;
```

- 위의 연산에서 salary가 null인 값은 무시됨
- count(*)를 제외한 모든 집계 함수에서는 입력 값 널을 무시함

## 3.8 중첩 하위 질의

- 다른 질의 안에 중첩된 select-from-where

### 3.8.1 집합 멤버십

- in, not in 은 집합 멤버십을 테스트

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
    course_id in (select course_id
                                from section
                                where semester = 'Spring' and
```

- (칼럼 1, 칼럼 2,..., 칼럼 N) in 절은 DBMS 벤더사마다 지원 여부 다름

### 3.8.2 집합 비교

```
select name
from instructor
where salary > some (select salary
                                            from instructor
                                            where dept_name = 'B


select name
```

```
from instructor
where salary > all (select salary
                                        from instructor
                                        where dept_name = 'Bi
```

- some(any), all 로 집합 원소에 대한 비교 가능

## 3.8.3 빈 릴레이션에 대한 테스트

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
    exists (select *
                    from section as T
                    where semester = 'Spring' and year = 2018
                                S.course_id = T.course_id);

select S.ID, S.name
from student as S
where not exists ((select course_id
                                    from course
                                    where dept_name = 'Biolo
                                    except
                                    (select T.course_id
                                    from takes as T
                                    where S.ID = T.ID));
```

- where 절의 하위 질의에서 바깥 질의에 있는 상관 이름을 사용할 수 있는 것을 상관 하위 질의라고 함
- 하위 질의에서 하위 질의나 하위 질의를 포함하는 질의의 상관 이름을 사용하는 것만 허용

## 3.8.4 중복 튜플이 없는지 테스트

```
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 a
                                            R.year = 2017);

select T.course_id
from course as T
where not unique (select R.course_id
                                        from section as R
                                        where T.course_id = R.cou
                                                    R.year = 2017
```

- 빈 집합인 경우 unique는 참으로 계산함
  - why? 공집합은 고유하니까?
- 릴레이션에 대한 unique는 t1 = t2인 두 개의 튜플을 포함하면 실패, 하나가 널 값이라면 t1 = t2 테스트는 실패
  - 하나가 널 값이라면 여러 개 다른 값이 있더라도 unique 는 참이 될 가능성이 있음

## 3.8.5 From 절의 하위 질의

```
select dept_name, avg_salary
from (select dept_name, avg(salary) as avg_salary
            from instructor
            group by dept_name)
where avg_salary > 42000;

select max (tot_salary)
from (select dept_name, sum(salary)
            from instructor
            group by dept_name) as dept_total (dept_name, tot
```

- MySQL과 PostgreSQL에서는 from 절의 각 하위 질의 릴레이션 이름을 절대 참조하지 않는다고 하더라도 반드시 해당 릴레이션의 이름을 지정해야 함
- from 절의 중첩된 하위 질의는 from 절에 있는 다른 릴레이션의 상관 변수를 사용할 수 없음

## 3.8.6 With 절

```
with max_budget (value) as
        (select max(budget)
        from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

- with 절은 해당 질의에서만 사용할 수 있는 임시 릴레이션 정의

## 3.8.7 스칼라 하위 질의

```
select dept_name,
                (select count(*)
                from instructor
                where department.dept_name = instructor.dept_
from department;
```

- 스칼라 하위 질의는 select, where , having 절에 나타날 수 있음
  - 집계 함수 없이 정의될 수 있음
  - 하나 이상의 튜플을 반환하면 오류 발생
- 스칼라 하위 질의의 결과는 릴레이션, 하지만 SQL은 암묵적으로 하나의 튜플의 하나의 속성으로부터 값 추출

## 3.8.3 From 절 없는 스칼라

```
(select count(*) from teaches) / (select count(*) from instru
```

# 3.9 데이터베이스의 변경

## 3.9.1 삭제

```
delete from r
where P;
```

```
delete from instructor
where salary < (select avg(salary)
                                from instructor);
```

- 하나의 릴레이션을 대상으로만 동작

## 3.9.2 삽입

```
insert into course
                values('CS-437', 'Database Systems', 'Comp. S

insert into course (title, course_id, credits, dept_name)
                values('CS-437', 'Database Systems', 'Comp. S

insert into instructor
             select ID, name, dept_name, 18000
             from student
             where dept_name = 'Music' and tot_cred > 144;
```

- 튜플의 속성값은 그 속성 도메인에 속해야 함
- 삽입될 튜플은 정확한 수의 속성을 가져야 함
- 삽입될 속성 명시 가능
- 대량 삽입은 특정 DBMS 벤더사에서 지원하는 bulk loader 유틸리티로 가능

## 3.9.3 갱신

```
update instructor
set salary = salary * 1.05;

update instructor
set salary = salary * 1.05
where salary < 70000;

update instructor
set salary = salary * 1.05
where salary < (select avg(salary)
                                from instructor);
```

```
update instructor
set salary = case
                                    when salary <= 100000 the
                                    else salary * 1.03
                               end
```

# 얘기해보면 좋을 것들

- 물리적인 행 수가 많은 레코드들을 삽입할 때 효율적인 방법?
  - 대용량 데이터 인강(샤딩)
  - Real MySQL