3. SQL

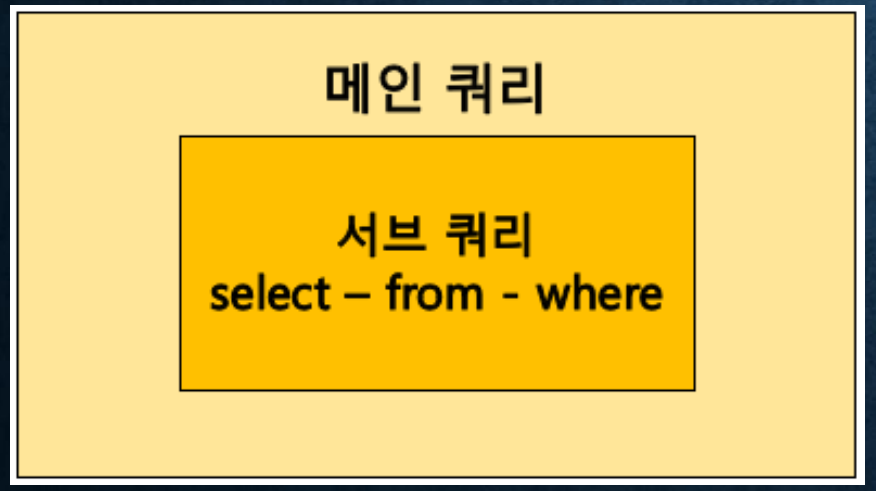
Nested Subqueries

Generated by 이유림

참고 강의 : https://www.edwith.org/database/lecture/27858/

NESTED SUBQUERY

- **Nested Subquery** : 쿼리 안에 또 쿼리
- 더 복잡한 표현 가능
- 간편함 : 여러 쿼리를 써서 표현할 것을 하나의 쿼리문으로 표현 가능



(대략 이런 구조)

- set membership / comparisons / cardinality 확인에 사용

SET MEMBERSHIP

: 릴레이션의 튜플이 어떤 집합의 멤버인지 아닌지 확인할 때 사용

- 연결사 2가지 : 해당 튜플이 set membership인지 테스트

(set : select 절에 의해 만들어진 값들의 집합)

1. IN

: 서브쿼리의 결과값 중 일치하는 것이 있으면 WHERE 절은 참

2. NOT IN

: 일치하는 것이 없으면 WHERE 절은 참

- IN, NOT IN은 enumerated set(열거형 집합)에도 적용 가능함

SET MEMBERSHIP – 속성1가지 예시1

예) *instructor*의 이름이 Mozart, Einstein 중 하나인지 테스트 – **NOT IN** 사용

```
SELECT DISTINCT name
FROM instructor
WHERE name NOT IN ('Mozart', 'Einstein') ;
// NOT IN 사용 : Mozart, Einstein 둘 다 아닌 경우 true
```

SET MEMBERSHIP – 속성1가지 예시2

예) 2009년 가을과 2010년 봄학기에 둘 다 개설되는 강의 찾기 – **집합연산(교집합)** 사용

```
SELECT DISTINCT course_id
FROM   section
WHERE semester='Fall' AND year=2009 AND
        course_id IN (   SELECT course_id
                         FROM section
                         WHERE semester='Spring' AND year=2010 ) ;
```

// **IN** : 메인쿼리의 course_id가 서브쿼리의 course_id에 **포함 O** → 원하는 결과

SET MEMBERSHIP – 속성1가지 예시3

예) 2009년 가을학기에는 있지만, 2010년 봄학기에는 없는 강의 찾기

```
SELECT DISTINCT course_id
FROM section
WHERE semester='Fall' AND year=2009 AND
        course_id NOT IN ( SELECT course_id
                           FROM section
                           WHERE semester='Spring' AND year=2010 ) ;
// NOT IN : 메인쿼리의 course_id가 서브쿼리의 course_id에 포함 X → 원하는 결과
```

SET MEMBERSHIP – 속성 여러개 예시

예) ID가 10101인 교수님의 수업을 듣는 학생들의 인원수 구하기

```
SELECT COUNT ( DISTINCT ID )
FROM tekes        // takes 릴레이션 = 학생의 수강 정보
WHERE (course_id, sec_id, semester, year) IN // 서브쿼리에서 10101교수님 수업 찾기
                ( SELECT course_id, sec_id, semester, year
                  FROM teaches                    // teaches 릴레이션 = 수업 정보
                  WHERE teaches.ID=10101 ) ;    // ID가 10101인 튜플 걸러내기
```

과정) 1. 서브쿼리에서 ID가 10101인 교수님의 수업 정보 구하기

2. 메인쿼리에서 해당 수업을 수강하는 학생 찾기

3. 중복 제거(DISTINCT) 후 학생 수 구하기 (여러 번 수강하는 경우 제외)

SET COMPARISON

: 메인쿼리의 결과와 서브쿼리의 결과를 비교할 때 사용

- **SOME 절** : 서브쿼리 결과값의 일부를 대신함 (비교한 결과 중 일부가 참이면 true)

  사용법 = 메인쿼리 비교(=,<,>등) **SOME** (서브쿼리)

- **ALL 절** : 서브쿼리 결과값의 모두를 대신함 (비교한 결과 중 모두가 참이면 true)

  사용법 = 메인쿼리 비교 **ALL** (서브쿼리)

SET COMPARISON – SOME절 예시1

예) 생물학과의 어떤 교수님(최소 한명)보다 salary가 많은 교수님들 이름 찾기

1. **SOME** 절 사용 X

```
SELECT  name
FROM    instructor AS T, instructor AS S
WHERE   T.salary > S.salary AND S.dept_name = 'Biology';
```

SET COMPARISON – SOME절 예시2

예) 생물학과의 어떤 교수님(최소 한명)보다 salary가 많은 교수님들 이름 찾기

2. Set Comparison – **SOME** 절 사용

```
SELECT  name
FROM    instructor
WHERE   salary > SOME (SELECT  salary
                       FROM    instructor
                       WHERE   dept_name='Biology' ) ;
```

SET COMPARISON – ALL절 예시

예) 생물학과의 모든 교수님보다 salary가 많은 교수님들 이름 찾기

```
SELECT  name
FROM    instructor
WHERE   salary > ALL  ( SELECT  salary
                        FROM    instructor
                        WHERE   dept_name='Biology' ) ;
```

SOME절 정의

- **F <comp> SOME r** (F: 비교대상, <comp>: 비교연산자)

  :"연산식을 만족하는 튜플 t가 릴레이션 r에 존재한다"

  (서브쿼리의 결과와 비교한 결과 중 하나라도 참이면, where절이 true

5 < SOME | 0 | 5 | 6 | (→ true)

5<0, 5<5는 거짓이지만, 5<6은 참 → true

5 < SOME | 0 | 5 | (→ false)

5<0, 5<5 모두 거짓 → false 모두 거짓일 때만 false

5 = SOME | 0 | 5 | (→ true)

5=0은 거짓이지만, 5=5는 참 → true

5 ≠ SOME | 0 | 5 | (→ true)

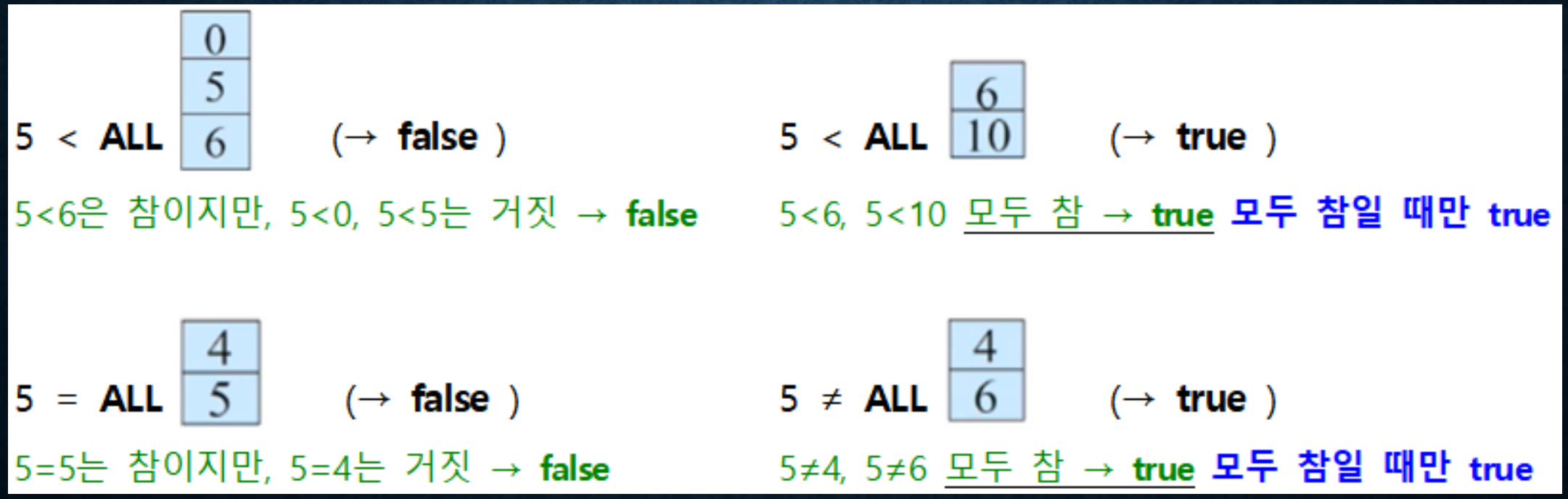
5≠5는 거짓이지만, 0≠5는 참 → true

ALL절 정의

- **F <comp> ALL r** (F: 비교대상, <comp>: 비교연산자)

  :"릴레이션 r의 모든 튜플은 연산식을 만족한다"

  (서브쿼리의 결과와 비교한 결과가 모두 참이면, where절이 true



SET COMPARISON과 SET MEMBERSHIP 비교

- **= SOME** : 여러 값 중 하나만이라도 같으면 됨 (**IN**과 같은 의미)
- **≠ SOME** : 여러 값 중 하나만이라도 다르면 됨
- **NOT IN** : 모두 달라야 함 = 일치하는 튜플이 하나라도 있으면 안 됨

  = SOME 과 **IN** 은 같은 의미지만, **≠ SOME** 과 **NOT IN** 은 다름

- **≠ ALL** : 모든 튜플이 조건에 맞지 않아야 함 (**NOT IN**과 같은 의미)
- **= ALL** : 모든 튜플이 비교 대상과 같아야 함
- **IN** : 일치하는 튜플이 하나라도 있으면 됨

  ≠ ALL 과 **NOT IN** 은 같은 의미지만, **= ALL** 과 **IN** 은 다름

EXISTS : 빈 릴레이션 검사

: 서브쿼리의 결과 릴레이션이 비었는지 아닌지 확인

- **EXISTS** : 인자로 들어온 서브쿼리의 결과가 비어있는지 확인
  : 조건을 만족하는 어떤 튜플이 존재하면(비어있지 않으면) true
- **NOT EXISTS** : 서브쿼리의 결과가 비어있을 때 true

- 요약) **EXISTS**는 안 비었으면(R≠Ø) TRUE
  NOT EXISTS는 비었으면(R=Ø) TRUE

CORRELATION VARIABLES (1)

Correlation Variable(name) : 릴레이션에 새 이름을 주기 위해 사용되는 식별자

예) 2009년 가을학기와 2010년 봄학기 모두 열리는 수업들 찾기 – **EXISTS** 사용

```
SELECT  course_id
FROM    section AS S
WHERE   semester='Fall' AND year=2009 AND
                EXISTS (SELECT  *
                        FROM    section AS T
                        WHERE   semester='Spring' AND year=2010
                                AND S.course_id = T.course_id ) ;
```

- 위의 쿼리에서 *S, T* 가 Correlation name임
- ■ 부분= 서브쿼리에서 Correlation valuable을 사용
- **EXISTS** 뒤에 나오는 부분= Correlation 서브쿼리 (Correlation valuable을 사용하는 서브쿼리)

CORRELATION VARIABLES (2)

예) 2009년 가을학기와 2010년 봄학기 모두 열리는 수업들 찾기 – **EXISTS** 사용

```
SELECT  course_id
FROM    section AS S
WHERE   semester='Fall' AND year=2009 AND
                EXISTS ( SELECT  *
                         FROM    section AS T
                         WHERE   semester='Spring' AND year=2010
                                 AND S.course_id = T.course_id ) ;
```

1. 서브 쿼리 = 2010년 봄학기의 수업 찾기
2. 메인 쿼리 = 2009년 가을학기의 수업 찾기
3. (서브쿼리 결과) == (메인쿼리 결과) 인 것을 쿼리의 최종 결과로 가져옴

NOT EXISTS 활용 예 (1)

릴레이션의 포함 여부 ($X \subseteq Y$) : $X-Y = \emptyset \leftrightarrow X \subseteq Y$

예) 생물학과에서 개설된 모든 수업을 수강하는 학생들 구하기

```
SELECT  DISTINCT S.ID, S.name
FROM    student AS S
WHERE   NOT EXISTS ( ( SELECT course_id // 서브쿼리1: 생물학과 개설 강의 찾기
                       FROM course
                       WHERE dept_name='Biology' ) EXCEPT // EXCEPT:서브1-서브2
                     ( SELECT  T.course_id // 서브쿼리2: 수강과목=생물학과수업 비교
                       FROM    takes AS T  // takes 릴레이션 = 학생의 수강 정보
                       WHERE   S.ID= T.ID ) ) ;
```

NOT EXISTS 활용 예 (2)

쿼리 해석)

WHERE 절 구조 = **NOT EXISTS** (서브쿼리1 **EXCEPT** 서브쿼리2)

1. 첫 번째 서브쿼리 = 생물학과에서 개설되는 수업의 course_id 찾기
2. 두 번째 서브쿼리 = 각 학생들이 수강한 수업의 course_id 구하기
   : 학생들의 ID(*S*.ID)와 수강한 학생의 ID(*T*.ID) 비교
3. 서브쿼리1 **EXCEPT** 서브쿼리2 (생물학과 수업 – 학생이 수강하는 수업)
   : 해당 학생이 생물학과의 모든 수업을 듣는다면, 결과릴레이션= ∅

UNIQUE : 중복된 튜플 확인

: 결과에 중복된 튜플이 있는지 없는지 확인

(중복 X → true, 서브쿼리가 빈 결과(공집합) 반환 → true)

예) 2009년에 많아야 한 번 개설된 강좌들 찾기

```
SELECT  T.course_id
FROM    course AS T      // section 릴레이션 = 수업의 이름과 개설학과
WHERE   UNIQUE ( SELECT R.course_id
                 FROM section AS R   // course 릴레이션 = 수업의 개설 정보
                 WHERE T.course_id = R.course_id AND R.year=2009 );
```

UNIQUE 예시 쿼리 해석

쿼리 해석)

1. 서브쿼리 = 2009년에 개설된 모든 수업 반환

2. 서브쿼리가 반환한 결과가 1개 또는 0개인 강좌는 **UNIQUE**에 의해 true

3. 2개 이상 반환된 강좌는 **UNIQUE**에 의해 false

FROM절에서의 SUBQUERY 1

- FROM 절에서 서브쿼리 표현 가능 (서브쿼리가 결과로 릴레이션을 반환하기 때문)

예) salary 평균이 42,000을 넘는 학과 교수들의 평균 salary 구하기 – **FROM절** 서브쿼리

```
SELECT dept_name, avg_salary        4. 결과:원하는 조건을 충족하는 학과 이름+평균 급여
FROM ( SELECT dept_name, AVG (salary) AS avg_salary   2. 학과별 salary 평균 구함
         FROM instructor
         GROUP BY dept_name )       1. 학과별로 그룹화
WHERE avg_salary>42000 ;            3. 서브쿼리의 결과로부터 salary평균>42,000만 고르기
```

FROM절에서의 SUBQUERY 2

예) salary 평균이 42,000을 넘는 학과 교수들의 평균 salary 구하기 – **FROM절** 서브쿼리

```
SELECT dept_name, avg_salary
FROM ( SELECT dept_name, AVG (salary)
       FROM instructor
       GROUP BY dept_name )
       AS dept_avg (dept_name, avg_salary)
WHERE avg_salary>42000 ;
```

- 설명)

  - FROM 절에서 사용된 서브쿼리의 결과 릴레이션을 'dept_avg'라고 이름 지음
  - 서브쿼리 결과 릴레이션의 속성 중 dept_name은 'dept_name'으로 유지
  - 속성 중 AVG (salary)의 결과는 'avg_salary'라고 이름 지음

FROM절에서의 SUBQUERY 3

- **LATERAL**을 앞에 붙인 서브쿼리는 FROM절의 서브쿼리, 이전 테이블의 속성에 접근 O

예) 각 교수의 이름, 급여, 소속학과의 평균 출력하기

```
SELECT  name, salary, avg_salary
FROM    instructor I1,   // 원래는 FROM절에서의 서브쿼리 내에서 I1 을 사용할 수 없음
        LATERAL ( SELECT AVG (salary) AS avg_salary      // →LATERAL : 사용 O
                  FROM instructor I2
                  WHERE I2.dept_name=I1.dept_name );    // 여기에서 I1 사용함
```

LATERAL절이 없다면, FROM절의 서브쿼리는 외부쿼리의 correlation variable에 접근 X

참고) LATERAL 키워드는 SQL 표준이지만, 많은 DB 시스템에서 지원 안 함