Chapter 5. 집계와 서브쿼리

20강. 행 개수 구하기 - COUNT

대표적인 집계함수: COUNT(), SUM(), AVG(), MIN(), MAX()

1) COUNT로 행 개수 구하기

집계함수

- SQL은 집합을 다루는 집계 함수 제공
- 인수로 집합을 지정하고 특정 방법으로 계산하여 그 결과 반환
- 복수의 값(집합)에서 하나의 값을 계산해님

  →SELECT수에 집계함수를 쓰면 WHERE구의 유무와 관계없이 결과값은 하나의 행

COUNT 함수

- 인수로 주어진 집합의 '개수'를 구해 반환
- 예제 5-1. sample51의 행 개수 구하기

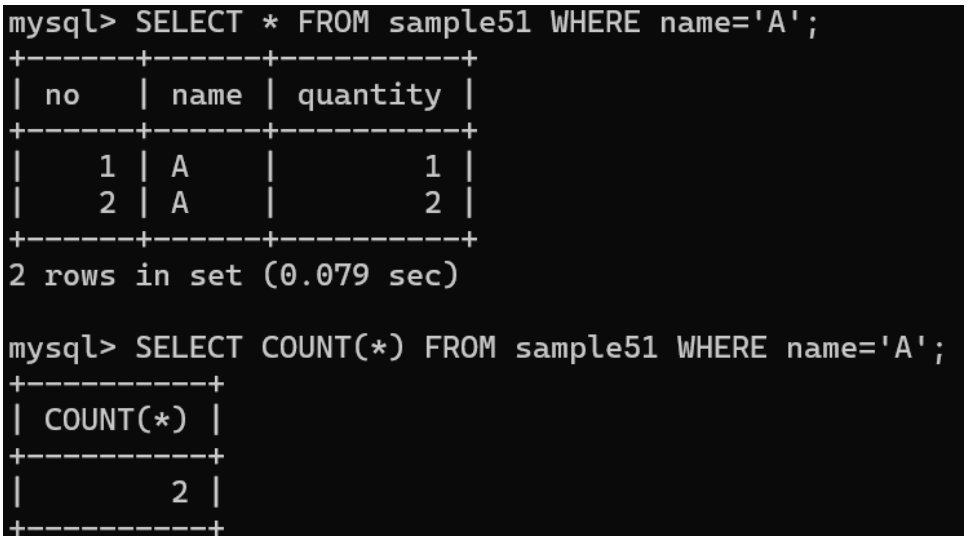
```
mysql> SELECT * FROM sample51;
+------+------+----------+
| no   | name | quantity |
+------+------+----------+
|    1 | A    |        1 |
|    2 | A    |        2 |
|    3 | B    |       10 |
|    4 | C    |        3 |
|    5 | NULL |     NULL |
+------+------+----------+
5 rows in set (0.827 sec)

mysql> SELECT COUNT(*) FROM sample51;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
```

  - 인수로 * 지정 : SELECT구의 모든 열을 나타낼 때 사용하는 메타 문자

    ⇒ 즉 테이블 전체를 가리킴, 테이블의 개수를 계산하는 것

WHERE 구 지정하기

- 예제 5-2. sample51의 행 개수를 WHERE구를 지정하여 구하기



```
mysql> SELECT * FROM sample51 WHERE name='A';
+------+------+----------+
| no   | name | quantity |
+------+------+----------+
|    1 | A    |        1 |
|    2 | A    |        2 |
+------+------+----------+
2 rows in set (0.079 sec)

mysql> SELECT COUNT(*) FROM sample51 WHERE name='A';
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

  - SELECT구는 WHERE 구보다 나중에 내부적으로 처리
  - WHERE 구의 조건에 맞는 행의 개수 구하기

2) 집계함수와 NULL값

COUNT의 인수로 열명 지정 가능

- 그 열에 한해서 행의 개수 구하기 가능
- **문제는 NULL값을 어떻게 취급하느냐?!**

  ⇒ 집계함수는 집합 안에 NULL값이 있을 경우 이를 제외하고 처리
- 예제 5-3. 행 개수를 구할 때 NULL 값 다루기

```
mysql> SELECT * FROM sample51;
+------+------+----------+
| no   | name | quantity |
+------+------+----------+
|    1 | A    |        1 |
|    2 | A    |        2 |
|    3 | B    |       10 |
|    4 | C    |        3 |
|    5 | NULL |     NULL |
+------+------+----------+
5 rows in set (0.010 sec)

mysql> SELECT COUNT(no), COUNT(name) FROM sample51;
+-----------+-------------+
| COUNT(no) | COUNT(name) |
+-----------+-------------+
|         5 |           4 |
+-----------+-------------+
```

- **다만 COUNT(*)의 경우, 모든 열의 행수를 카운트하기 때문에 NULL의 정보 무시X**

3) DISTINCT로 중복 제거

집합을 다룰 때, 경우에 따라 집합 안에 중복된 값이 있는지 여부가 문제가 될 수도 있다.

DISTINCT: 중복된 값을 제거하는 함수

- ALL: 중복 유무와 상관없이 문자 그대로 모든 행 반환
- ALL 또는 DISTINCT 지정 생략 → ALL로 간주

- sample51 테이블은 name열에 중복된 값이 있음
- 예제 5-5. DISTINCT로 중복 제거하기

```
mysql> SELECT DISTINCT name FROM sample51;
+------+
| name |
+------+
| A    |
| B    |
| C    |
| NULL |
+------+
```

4) 집계함수에서 DISTINCT

- DISTINCT는 집계함수의 인수에 수식자로 지정할 수 있음
- DISTINCT를 이용해 집합에서 중복을 제거한 뒤 COUNT로 개수를 셀 수 있음
- 예제 5-6. 중복을 제거한 뒤 개수 구하기

```
mysql> SELECT COUNT(ALL name), COUNT(DISTINCT name) FROM sample51;
+-----------------+---------------------+
| COUNT(ALL name) | COUNT(DISTINCT name) |
+-----------------+---------------------+
|               4 |                   3 |
+-----------------+---------------------+
```

21강. COUNT 이외의 집계함수

1) SUM으로 합계 구하기

SUM 집계함수: 집합의 합계를 구하는 집계 함수

- 지정되는 집합은 수치형 뿐, 문자열형 · 날짜시간형 집합의 합계는 불가능
- NULL값은 무시, NULL값 제거 후 합계
- 예제 5-7. SUM으로 합계 구하기

```
mysql> SELECT SUM(quantity) FROM sample51;
+---------------+
| SUM(quantity) |
+---------------+
|            16 |
+---------------+
```

2) AVG로 평균내기

평균값 내는 방법

1. SUM집계함수의 결과값에 개수 나누기
2. SUM과 COUNT이용 : 예) SUM(quantity) / COUNT(quantity)
3. **AVG 집계함수** 이용하기 >>new!

AVG 집계함수

- 수치형만 가능
- NULL값 무시
- 예제 5-8. AVG로 평균값 구하기

```
mysql> SELECT AVG(quantity), SUM(quantity)/COUNT(quantity) FROM sample51;
+---------------+-------------------------------+
| AVG(quantity) | SUM(quantity)/COUNT(quantity) |
+---------------+-------------------------------+
|        4.0000 |                        4.0000 |
+---------------+-------------------------------+
```

- NULL을 0으로 간주하여 평균→CASE를 사용해 NULL을 0으로 변환 후 AVG이용
- 예제 5-9. AVG로 평균값 계산 (NULL을 0으로 변환)

```
mysql> SELECT AVG(CASE WHEN quantity IS NULL THEN 0 ELSE quantity END) AS avgnull0 FROM
 sample51;
+----------+
| avgnull0 |
+----------+
|   3.2000 |
+----------+
```

3) MIN · MAX로 최솟값 · 최댓값 구하기

MIN 집계함수, MAX 집계함수

각각 최솟값과 최댓값을 구하는 집계함수

- 문자열형과 날짜시간형에도 사용가능
- NULL 값 무시
- 예제 5-10. MIN, MAX로 최솟값, 최댓값 구하기

```
mysql> SELECT MIN(quantity), MAX(quantity), MIN(name), MAX(name) FROM sample51;
+---------------+---------------+-----------+-----------+
| MIN(quantity) | MAX(quantity) | MIN(name) | MAX(name) |
+---------------+---------------+-----------+-----------+
|             1 |            10 | A         | C         |
+---------------+---------------+-----------+-----------+
1 row in set (0.125 sec)
```

22강. 그룹화 - GROUP BY

1) GROUP BY로 그룹화

- 예제 5-11. sample51테이블

```
mysql> SELECT * FROM sample51;
+------+------+----------+
| no   | name | quantity |
+------+------+----------+
|    1 | A    |        1 |
|    2 | A    |        2 |
|    3 | B    |       10 |
|    4 | C    |        3 |
|    5 | NULL |     NULL |
+------+------+----------+
5 rows in set (0.014 sec)
```

  - name열에 같은 값을 가진 행끼리 한데 묶어 그룹화한 집합을 집계함수로 넘길 수 있음

    ⇒ **GROUP BY**사용

- 예제 5-12. name열로 그룹화하기

```
mysql> SELECT name FROM sample51 GROUP BY name;
+------+
| name |
+------+
| A    |
| B    |
| C    |
| NULL |
+------+
4 rows in set (0.072 sec)
```

  - DISTINCT를 사용했을 때와 같은 결과값
  - but, DISTINCT의 효과가 있는 것이고 내부적인 과정은 다름

    (같은 행이 하나의 그룹으로 묶이는 것)

- 예제 5-13. name 열을 그룹화해서 계산하기

```
mysql> SELECT name, COUNT(name), SUM(quantity) FROM sample51 GROUP BY name;
+------+-------------+---------------+
| name | COUNT(name) | SUM(quantity) |
+------+-------------+---------------+
| A    |           2 |             3 |
| B    |           1 |            10 |
| C    |           1 |             3 |
| NULL |           0 |          NULL |
+------+-------------+---------------+
```

- GROUP BY는 업무 환경에서 쓰이는 경우가 많음
  - 예를 들어 점포별, 상품별, 월별, 일별 등 특정 단위로 집계할 때 많이 사용

2) HAVING 구로 조건 지정

집계함수에서는 WHERE 구의 조건식을 사용할 수 없음!

- WHERE 구로 행을 검색하는 처리가 GROUP BY로 그룹화 처리 보다 앞선 순서

  ⇒ **HAVING 구를 사용하여 집계함수 조건식 지정!**

HAVING 구

- GROUP BY 구의 뒤에 기술
- WHERE 구와 동일하게 조건식 지정 가능
- 조건식에는 그룹별로 집계된 열의 값이나 집계함수의 계산결과가 전달
- 조건식이 참인 그룹값만 클라이언트에게 반환
- 예제 5-14. HAVING을 사용해 검색

```
mysql> SELECT name, COUNT(name) FROM sample51
    -> GROUP BY name HAVING COUNT(name)=1;
+------+-------------+
| name | COUNT(name) |
+------+-------------+
| B    |           1 |
| C    |           1 |
+------+-------------+
2 rows in set (0.139 sec)
```

📌 **내부 처리 순서**

WHERE 구 → GROUP BY 구 → HAVING 구 → SELECT 구 → ORDER BY 구

- 다만 HAVING구는 SELECT구보다 먼저 처리 되기 때문에 별명 사용 불가능

3) 복수열의 그룹화

GROUP BY에 사용 시 주의사항

> GROUP BY에 지정한 열 이외의 열은 집계함수를 사용하지 않은 채 SELECT 구에 기술 X

```
#SELECT구에 기술할 수 없는 열
SELECT no, name, quantity FROM sample51 GROUP BY name;
```

- no, quantity 열과 name열의 반환 되는 개수가 다름에 따라 에러가 생김

```
#다음과 같이 쿼리 수정
SELECT MIN(no), name, SUM(quantity) FROM sample51 GROUP BY name;
```

4) 결괏값 정렬

- GROUP BY로 그룹화해도 실행결과 순서 정렬은 불가능
- ORDER BY를 사용해 결과 정렬
- 예제 5-15. 집계한 결과 정렬하기

```
mysql> SELECT name, COUNT(name), SUM(quantity) FROM sample51
    -> GROUP BY name ORDER BY SUM(quantity) DESC;
+------+-------------+---------------+
| name | COUNT(name) | SUM(quantity) |
+------+-------------+---------------+
| B    |           1 |            10 |
| A    |           2 |             3 |
| C    |           1 |             3 |
| NULL |           0 |          NULL |
+------+-------------+---------------+
4 rows in set (0.148 sec)
```

23강. 서브쿼리

서브쿼리

SELECT 명령에 의한 데이터 질의로, 상부가 아닌 하부의 부수적인 질의를 의미

- SQL 명령문 안에 하부 명령을 괄호로 묶어 지정
- WHERE 구에서 많이 사용

1) DELETE의 WHERE구에서 서브쿼리 사용하기

- 예제 5-16. sample54 테이블

```
mysql> SELECT * FROM sample54;
+------+------+
| no   | a    |
+------+------+
|    1 |  100 |
|    2 |  900 |
|    3 |   20 |
|    4 |   80 |
```

  - a열의 값이 가장 작은 행을 삭제 하고자 함 → 예제 5-18

- 예제 5-17. sample54에서 a의 최솟값 검색하기

```
mysql> SELECT MIN(a) FROM sample54;
+--------+
| MIN(a) |
+--------+
|     20 |
+--------+
```

- 예제 5-18. 최솟값을 가지는 행 삭제하기

- 예제 5-17과 5-16을 한번에!(최솟값 찾기+삭제 를 한번에)

```
DELETE FROM sample54 WHERE a = (SELECT MIN(a) FROM sample5
4);
```

- 다만 예제 5-18. 실행문은 MySQL에서 실행하면 오류가 남
  - 데이터를 추가/갱신할 경우 동일한 테이블을 서브쿼리로 사용할 수 없도록 돼있음

```
#MySQL에서 에러 없이 갱신하는 방법: 인라인 뷰로 임시 테이블 만들기
DELETE FROM sample54 WHERE a=(SELECT a FROM (SELECT MIN
(a) AS a FROM sample54) AS x);
```

    - 클라이언트 변수 (MySQL에 한해서)

      @a가 변수, set이 변수에 대입하는 명령

```
set @a = (SELECT MIN(a) FROM sample54);
DELETE FROM sample54 WHERE a=@a;
```

2) 스칼라 값

- 예제 5-19. 서브쿼리의 패턴
  - 패턴 1. 하나의 값을 반환하는 패턴

```
mysql> SELECT MIN(a) FROM sample54;
+--------+
| MIN(a) |
+--------+
|     20 |
+--------+
```

  - 패턴 2. 복수의 행이 반환되지만 열은 하나인 패턴

```
mysql> SELECT no FROM sample54;
+------+
| no   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
```

- 패턴 3. 하나의 행이 반환되지만 열이 복수인 패턴

```
mysql> SELECT MIN(a), MAX(no) FROM sample54;
+--------+---------+
| MIN(a) | MAX(no) |
+--------+---------+
|     20 |       4 |
+--------+---------+
```

- 패턴 4. 복수의 행, 복수의 열이 반환되는 패턴

```
mysql> SELECT no, a FROM sample54;
+------+------+
| no   | a    |
+------+------+
|    1 |  100 |
|    2 |  900 |
|    3 |   20 |
|    4 |   80 |
+------+------+
```

- [예제 5-19]에서 첫 번째 패턴만 다름

  → 하나의 값만 반환

  → 단일 값으로도 통용되지만 데이터베이스 업계에서는 **'스칼라 값'**이라 불림
- SELECT명령에서 스칼라 값은 서브쿼리로 사용하기 용이
  - 스칼라 값을 반환하도록 SELECT명령을 작성할 땐 SELECT구에 단일 열 지정
  - WHERE 구에서 스칼라 값을 반환하는 서브쿼리는 **=연산자**로 비교 가능
- 스칼라 서브쿼리
  - 스칼라 값을 반환하는 서브쿼리
  - 스칼라 서브쿼리라면 WHERE구에서도 집계함수 사용 가능

3) SELECT구에서 서브쿼리 사용하기

- 문법적으로 서브쿼리는 하나의 항목으로 취급
- 문법적으로 문제는 없지만 실행하면 에러가 생기는 경우가 있음
  - 이는 스칼라 값의 반환 여부에 따라 생기는 현상
  - 서브쿼리를 사용할 때는 스칼라 서브쿼리로 되어있는지 확인해야함
- SELECT 구에서 서브쿼리를 지정할 때, 스칼라 서브쿼리가 필요
- 예제 5-20. SELECT 구에서 서브쿼리 사용하기

```
mysql> SELECT
    -> (SELECT COUNT(*) FROM sample51) AS sq1,
    -> (SELECT COUNT(*) FROM sample54) AS sq2;
+------+------+
| sq1  | sq2  |
+------+------+
|    5 |    4 |
+------+------+
```

  - 여기서 주의할 점: 상부의 SELECT명령에 FROM 구가 없음

    MySQL은 FROM 구 생략 가능, but Oracle 등 전통적인 데이터베이스 제품은 생략 불가능

    Oracle에서는 FROM DUAL로 지정하면 실행

```
-- 예제 5-21. SELECT 구에서 서브쿼리 사용하기 (Oracle경우)
SELECT
   (SELECT COUNT(*) FROM sample51) AS sq1,
   (SELECT COUNT(*) FROM sample54) AS sq2 FROM DUAL;
```

4) SET구에서 서브쿼리 사용하기

- 예제 5-22. SET 구에서 서브쿼리 사용하기

```
UPDATE sample54 SET a=(SELECT MAX(a) FROM sample54);
```

- SET 구에서 서브쿼리를 사용할 경우 스칼라 값을 반환하도록 스칼라 서브쿼리를 지정할 필요가 있음

5) FROM구에서 서브쿼리 사용하기

- FROM 구에 기술할 경우, 스칼라 값을 반환하지 않아도 좋음
- 예제 5-23. FROM 구에서 서브쿼리 사용하기

```
mysql> SELECT * FROM (SELECT * FROM sample54) sq;
+------+------+
| no   | a    |
+------+------+
|    1 |  100 |
|    2 |  900 |
|    3 |   20 |
|    4 |   80 |
+------+------+
```

  - nested구조, 중첩구조, 내포구조 : SELECT명령 안에 SELECT명령이 있음
  - FROM구에서는 테이블이나 서브쿼리에 별명을 붙일 수 있음
- 예제 5-24. FROM 구에서 서브쿼리 사용하기(AS로 지정)

```
mysql> SELECT * FROM (SELECT * FROM sample54) AS sq;
+------+------+
| no   | a    |
+------+------+
|    1 |  100 |
|    2 |  900 |
|    3 |   20 |
|    4 |   80 |
+------+------+
```

  - AS 키워드를 이용해 서브쿼리의 별명(이름) 지정
  - 단, Oracle에서는 AS를 붙이면 에러 → AS 붙이지 않음
- 예제 5-25. FROM 구에서 서브쿼리 사용하기 (3단계)

```
mysql> SELECT * FROM (SELECT * FROM (SELECT * FROM sample54) sq1) sq2;
+------+------+
| no   | a    |
+------+------+
|    1 |  100 |
|    2 |  900 |
|    3 |   20 |
|    4 |   80 |
+------+------+
```

  - 중첩구조는 몇 단계로든 구성 가능
  - 그러나 이 예제처럼 테이블 한 개 지정하는데 3단계까지는 작성X

실제 업무에서 FROM 구에서 서브쿼리를 지정하여 사용하는 경우

FROM구에서 서브쿼리를 사용하는 것으로 Oracle의 행 제한 가능

```
-- 예제 5-26. Oracle에서 LIMIT 구의 대체 명령
SELECT * FROM (
   SELECT * FROM sample54 ORDER BY a DESC
) sq
WHERE ROWNUM <= 2;
```

6) INSERT 명령과 서브쿼리

- INSERT 명령과 서브쿼리 조합 방법
  1. VALUES 구의 일부로 서브쿼리 사용
  2. VALUES 구 대신 SELECT 명령 사용
- 예제 5-27. VALUES 구에서 서브쿼리 사용하기

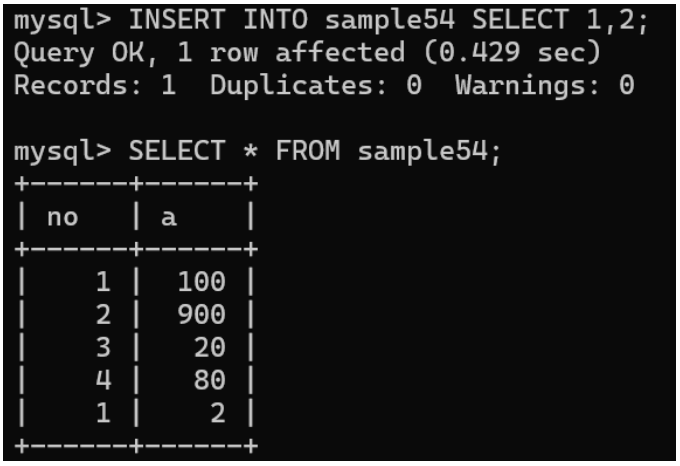
```
INSERT INTO sample54 VALUES(
   (SELECT COUNT(*) FROM sample51),
   (SELECT COUNT(*) FROM sample54)
);
SELECT * FROM sample54;
```

  - 서브 쿼리는 스칼라 서브쿼리로 지정
  - 자료형도 일치해야함

INSERT SELECT

VALUES 대신 SELCT 명령 사용 예

- 예제 5-28. SELECT 결과를 INSERT 하기



  - INSERT SELECT라 불리는 명령: INSERT + SELECT 합친 것 같은 명령
  - INSERT INTO sample54 VALUES(1,2)의 경우와 같음
  - SELECT명령이 반환하는 값이 스칼라 값일 필요는 없음
  - SELECT 반환 열 수, 자료형이 INSERT할 테이블과 일치하면 됨
  - 데이터의 복사나 이동 시 자주 사용
- 예제 5-29. 테이블의 행 복사하기

```
INSERT INTO sample542 SELECT * FROM sample543;
```

  - 열 구성이 똑같은 테이블 사이는 이러한 INSERT SELECT 명령으로 행 복사 가능

24강. 상관 서브쿼리

상관 서브쿼리는 서브 쿼리의 일종

1) EXISTS

EXISTS 술어

서브쿼리를 사용해 검색할 때 '데이터가 존재하는지 아닌피' 판별하기 위한 조건을 지정할 수 있음

- 서브쿼리가 반드시 스칼라 값일 필요 X
- 예제 5-30. sample551 테이블과 sample552테이블

```
mysql> SELECT * FROM sample551; SELECT * FROM sample552;
+------+------+
| no   | a    |
+------+------+
|    1 | NULL |
|    2 | NULL |
|    3 | NULL |
|    4 | NULL |
|    5 | NULL |
+------+------+
5 rows in set (0.010 sec)

+------+
| no2  |
+------+
|    3 |
|    5 |
+------+
```

  - sample551에 no열의 값과 같은 행이 있다면 '있음'이라는 값으로, 없다면 '없음'으로 갱신

    → 예제 5-31.

- 예제 5-31. EXISTS를 사용해 '있음'으로 갱신하기

```
mysql> UPDATE sample551 SET a ='있음' WHERE
    -> EXISTS (SELECT * FROM sample552 WHERE no2=no);
Query OK, 2 rows affected (0.285 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT * FROM sample551;
+------+------+
| no   | a    |
+------+------+
|    1 | NULL |
|    2 | NULL |
|    3 | 있음 |
|    4 | NULL |
|    5 | 있음 |
+------+------+
```

2) NOT EXISTS

NOT EXISTS

'없음'의 경우, 행이 존재하지 않는 상태가 참

- 예제 5-32. NOT EXISTS를 사용해 '없음'으로 갱신하기

```
mysql> UPDATE sample551 SET a='없음' WHERE
    -> NOT EXISTS (SELECT * FROM sample552 WHERE no2=no)
Query OK, 3 rows affected (0.155 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM sample551;
+------+------+
| no   | a    |
+------+------+
|    1 | 없음 |
|    2 | 없음 |
|    3 | 있음 |
|    4 | 없음 |
|    5 | 있음 |
+------+------+
```

3) 상관 서브쿼리

상관 서브쿼리

부모 명령과 자식인 서브쿼리가 특정 관계를 맺는 것

- 예제 5-31에서 UPDATE 명령(부모)에서는 sample551 갱신, 자식인 서브쿼리는 sample552테이블의 no2 열 값이 부모의 no 열 값과 일치하는 행을 검색
- 부모 명령과 연관되어 처리되기 때문에 서브 쿼리 부분만 따로 실행 불가

테이블명 붙이기

만약 다른 테이블에 동일한 열의 이름이 있다면 동작하지 어려움

- 테이블 지정을 하여 정상적 처리가 되도록 함
- no 열이 sample551의 것이라면 ‘sample551.no’으로 지정
- 예제 5-33. 열에 테이블명 붙이기

```
UPDATE sample551 SET a='있음' WHERE
   → EXISTS (SELECT * FROM sample552 WHERE sample552.no2=sam
ple551.no);
```

4) IN

IN

집합 안의 값이 존재하는지 조사할 수 있다

- 예제 5-34. IN을 사용해 조건식 기술

```
mysql> SELECT * FROM sample551 WHERE no IN(3, 5);
+------+------+
| no   | a    |
+------+------+
|    3 | 있음 |
|    5 | 있음 |
+------+------+
2 rows in set (0.132 sec)
```

- 예제 5-35. IN의 오른쪽을 서브쿼리로 지정하기
  - 집합 부분은 서브쿼리로 지정할 수 있음

```
mysql> SELECT * FROM sample551 WHERE no IN
    -> (SELECT no2 FROM sample552);
+------+------+
| no   | a    |
+------+------+
|    3 | 있음 |
|    5 | 있음 |
+------+------+
2 rows in set (0.135 sec)
```

IN과 NULL

IN에서는 집합 안에 NULL값이 있어도 무시 X

- 다만 NULL = NULL은 계산할 수 없음
- NULL을 비교할 때는 **IS NULL**사용
- NOT IN의 경우, 집합 안에 NULL값이 있으면 참을 반환 X

  → 결과는 '불명(UNKNOWN)'이 된다