

5장 집계와 서브쿼리

2019.08.03

13기 박세민

행 개수 구하기 – COUNT

- COUNT()
- SUM()
- AVG()
- MIN()
- MAX()
- 보통 함수들은 한 행에 대해 하나의 값을 반환한다.
- 그런데, 집계함수들은 인자로 집합을 받는다
- ex SELECT COUNT(*) FROM *table_name*;

행 개수 구하기 – COUNT

- COUNT()
- COUNT(*) 대신 COUNT(*col_name*)
을 쓰면 해당 열에 대해 NULL 이
아닌 행의 개수를 반환한다.

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x27
[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 (0.00 sec)

[mysql> SELECT COUNT(*) FROM sample51;
+-----+
| COUNT(*) |
+-----+
| 5         |
+-----+
1 row in set (0.00 sec)

mysql> |
```

행 개수 구하기 – COUNT

- COUNT()
- WHERE 와 함께 사용할 수도 있다.
- Ex) SELECT COUNT(*) FROM sample51 where name = 'A';

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x27
[mysql> SELECT * FROM sample51 WHERE name = 'A';
+-----+-----+
| no  | name | quantity |
+-----+-----+
| 1  | A    | 1        |
| 2  | A    | 2        |
+-----+-----+
2 rows in set (0.00 sec)

[mysql> SELECT COUNT(*) FROM sample51 WHERE name = 'A';
+-----+
| COUNT(*) |
+-----+
| 2        |
+-----+
1 row in set (0.00 sec)

mysql> |
```

행 개수 구하기 - COUNT

- SELECT 이후에 DISTINCT 예약어를 붙이면 중복을 제거해준다.
- 이를 사용해 고유한 값들의 개수를 구할 수도 있다
- SELECT COUNT(DISTINCT name)
FROM sample51;

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x27
[mysql> SELECT DISTINCT name FROM sample51;
+-----+
| name |
+-----+
| A     |
| B     |
| C     |
| NULL  |
+-----+
4 rows in set (0.00 sec)

[mysql> SELECT COUNT(DISTINCT name) FROM sample51;
+-----+
| COUNT(DISTINCT name) |
+-----+
| 3                     |
+-----+
1 row in set (0.00 sec)

mysql> |
```

COUNT 이외의 집계함수

- SUM([ALL | DISTINCT] 집합)
- AVG([ALL | DISTINCT] 집합)
- MIN([ALL | DISTINCT] 집합)
- MAX([ALL | DISTINCT] 집합)
- ALL 혹은 DISTINCT 예약어를 통해 전체 혹은 고유한 값들에 대해 합, 평균, 최소, 최대 등의 집계를 낼 수 있다.
- 집계함수들은 NULL 값을 무시한다. 만약 NULL을 0으로 바꿔 처리하고 싶다면 CASE를 사용해 NULL을 0으로 변환한 뒤에 계산하면 된다.

COUNT 이외의 집계함수

- SUM([ALL | DISTINCT] 집합)
- Ex) SELECT SUM(quantity) FROM sample51;
- 위의 경우 SUM(**ALL** quantity)와 같다.
- DISTINCT 예약어를 생략하면 ALL을 기본으로 사용한다.

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x27
[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 (0.00 sec)

[mysql> SELECT SUM(quantity) FROM sample51;
+-----+
| SUM(quantity) |
+-----+
| 16            |
+-----+
1 row in set (0.01 sec)

mysql> |
```

COUNT 이외의 집계함수

- AVG([ALL | DISTINCT] 집합)
- Ex) SELECT AVG(quantity), SUM(quantity)/COUNT(quantity) FROM sample51;

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x21
[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 (0.00 sec)

[mysql> SELECT AVG(quantity), SUM(quantity)/COUNT(quantity) FROM sample51;
+-----+-----+-----+
| AVG(quantity) | SUM(quantity)/COUNT(quantity) |
+-----+-----+-----+
| 4.0000        | 4.0000                          |
+-----+-----+-----+
1 row in set (0.02 sec)

mysql> |
```


COUNT 이외의 집계함수

- MIN([ALL | DISTINCT] 집합)
- MAX([ALL | DISTINCT] 집합)
- Ex) SELECT MIN(quantity), MAX(quantity), MIN(name), MAX(name) FROM sample51;

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x21
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 (0.00 sec)

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 (0.01 sec)

mysql> |
```

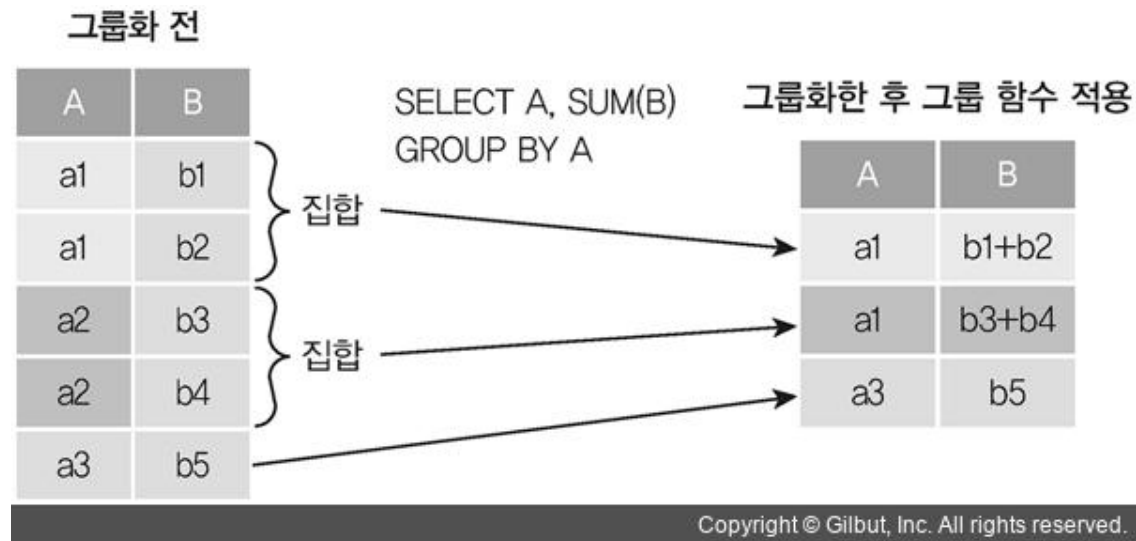
COUNT 이외의 집계함수

- CASE 를 사용해 NULL 값 처리하기
- Ex) SELECT AVG(CASE WHEN quantity IS NULL THEN 0 ELSE quantity END) FROM sample51;

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x21
[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 (0.00 sec)

mysql> SELECT AVG(CASE WHEN quantity IS NULL THEN 0 ELSE quantity END) FROM sample51;
+-----+
| AVG(CASE WHEN quantity IS NULL THEN 0 ELSE quantity END) |
+-----+
| 3.2000 |
+-----+
1 row in set (0.00 sec)
```

그룹화 – GROUP BY



- GROUP BY A를 먼저 한 후, 각 그룹에 대해 SUM(B)를 하는 것과 같은 결과

그룹화 – GROUP BY

- SELECT name, COUNT(name), SUM(quantity) FROM sample51 GROUP BY name;

```
SPark9625 — mysql -u root -p --local-infile=1 — 81x21
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          |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> |
```

그룹화 – GROUP BY

- 주의사항
- 내부처리 순서: WHERE --> GROUP BY --> SELECT --> ORDER BY
- 따라서 `SELECT name FROM sample51 WHERE COUNT(name) = 1 GROUP BY name;` 같은 쿼리는 불가능 (name으로 그룹화한 후 COUNT가 1이면 선택하고 싶은데, WHERE가 우선이라 안된다)
- 이런 경우에는 HAVING을 써야한다.

그룹화 – GROUP BY

- HAVING 예시
- SELECT name, COUNT(name) FROM sample51 GROUP BY name HAVING COUNT(name) = 1;

```
SPark9625 — mysql -u root -p --local-infile=1 — 91x21
[mysql> SELECT name, COUNT(name) FROM sample51 GROUP BY name;
+-----+-----+
| name | COUNT(name) |
+-----+-----+
| A     |          2 |
| B     |          1 |
| C     |          1 |
| NULL  |          0 |
+-----+-----+
4 rows in set (0.00 sec)

[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 (0.00 sec)

mysql> |
```

그룹화 – GROUP BY

- 내부처리 순서: WHERE --> GROUP BY --> HAVING --> SELECT --> ORDER BY
- 따라서 HAVING은 SELECT 보다 먼저 실행되므로, 별명을 사용할 수는 없다

서브쿼리

- 쿼리 안에 다른 쿼리를 쓰는것을 의미함
- ... (SELECT <명령>) ...



[그림 II-2-13] 단일 행 서브쿼리의 예제1

서브쿼리

- DELETE의 WHERE에서 서브쿼리 사용
- WHERE에서는 스칼라값과 비교를 하므로, 서브쿼리의 결과값도 스칼라값으로 나와야 한다.

```
SPark9625 — mysql -u root -p --local-infile=1 — 63x24
[mysql> SELECT * FROM sample54;
+-----+-----+
| no  | a  |
+-----+-----+
| 1  | 100 |
| 2  | 90  |
| 3  | 20  |
| 4  | 80  |
+-----+-----+
4 rows in set (0.00 sec)

[mysql> SELECT MIN(a) FROM sample54;
+-----+
| MIN(a) |
+-----+
| 20      |
+-----+
1 row in set (0.00 sec)

[mysql> DELETE FROM sample54 WHERE a = (
  -> SELECT a FROM (SELECT MIN(a) AS a FROM sample54) AS x
  -> );
Query OK, 1 row affected (0.01 sec)
```

서브쿼리

- SELECT에서 서브쿼리 사용
- 스칼라 서브쿼리 필요

```
SPark9625 — mysql -u root -p --local-infile=1 — 91x24
[mysql> SELECT
[   -> (SELECT COUNT(*) FROM sample51) AS sq1,
[   -> (SELECT COUNT(*) FROM sample54) AS sq2;
+-----+-----+
| sq1 | sq2 |
+-----+-----+
|  5  |  3  |
+-----+-----+
1 row in set (0.00 sec)

mysql> |
```

서브쿼리

- UPDATE의 SET에서 서브쿼리 사용

```
SPark9625 — mysql -u root -p --local-infile=1 — 91x23
[mysql> SELECT * FROM sample54;
+-----+-----+
| no    | a    |
+-----+-----+
| 1     | 100  |
| 2     | 90   |
| 4     | 80   |
+-----+-----+
3 rows in set (0.01 sec)

[mysql> UPDATE sample54 SET a = (SELECT a FROM (SELECT MAX(a) as a FROM sample54) AS x);
Query OK, 2 rows affected (0.01 sec)
Rows matched: 3  Changed: 2  Warnings: 0

[mysql> SELECT * FROM sample54;
+-----+-----+
| no    | a    |
+-----+-----+
| 1     | 100  |
| 2     | 100  |
| 4     | 100  |
+-----+-----+
3 rows in set (0.00 sec)
```

서브쿼리

- FROM에서 서브쿼리 사용
- FROM은 테이블을 가리키기 때문에
서브쿼리가 스칼라가 아니어도 상관없다
- (오라클에서는 LIMIT 대신 서브쿼리 +
WHERE ROWNUM <= N 으로 한다고...)

```
SPark9625 — mysql -u root -p --local-infile=1 — 65x23
mysql> SELECT * FROM (SELECT * FROM sample54) AS another_table;
+-----+-----+
| no  | a    |
+-----+-----+
| 1   | 100  |
| 2   | 90   |
| 4   | 80   |
+-----+-----+
3 rows in set (0.00 sec)

mysql> |
```

서브쿼리

- INSERT와 서브쿼리
- INSERT INTO sample541 VALUES (
 (SELECT COUNT(*) FROM sample51),
 (SELECT COUNT(*) FROM sample54)
);
- INSERT INTO sample541 SELECT 1,2;
- INSERT INTO sample541 (SELECT * FROM sample 542);

a	b
5	3

a	b
5	3
1	2

상관 서브쿼리

- 한 테이블이 아닌, 두 개의 테이블의 관계를 바탕으로 쿼리하는것.
- 일반적인 서브쿼리는 서브쿼리만 떼어내도 작동하지만, 상관 서브쿼리의 경우 서브쿼리만 돌리면 에러가 난다.
- Ex) UPDATE sample551 SET a = '있음' WHERE no IN (SELECT * FROM sample552 WHERE sample551.no = sample552.no);

no	a
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL

sample551

no
3
5

sample552



no	a
1	NULL
2	NULL
3	있음
4	NULL
5	있음

sample551

상관 서브쿼리

- 주의사항
- IN은 NULL 값은 무시한다.
- 따라서 NULL에 대해 처리를 하고 싶다면 IS NULL 을 사용해야 한다.