3주차 Basic SQL Multiset, Nested query, ... RDBMS

2022. 3. 20. 9:59

https://blog.naver.com/lsm980515/222677733373

SFW절의 역할을 알아보자

복습 차원에서 SFW절의 역할을 알아보자면

SELECT : 특정 attribute를 지정

FROM : 데이터를 검색할 table을 나타내는 절

WHERE : 데이터를 필터링 하는 역할을 하며 query에서 반환되는 row를 제한하는 비교 조건자가 포함됨.

※WHERE절에서 비교 조건자가 True로 평가되지 않은 결과 집합에서 모든 행을 제거함.

Multiset Operation

Multiset은 이전 post에서도 말했지만

동일 항목이 여러 개 출현하는 것을 허락한 집합체 이다 .

즉, **증복은 허용하며, 순서 또한 상관x**

또 DB에서 집합이라는 의미는 tuple의 집합을 말한다.

※ DB에서 모든 집합연산자는 중복을 허용하지 않지만 union all과 같이 중복을 허용하여 나타내는 방법이 있다.

Multiset X

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)

=

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

multiset을 표현하는 두가지 방법들이며

오른쪽 그림과 같이 표현하는 이유는 table의 길이가 길어 지기 때문이다.

λ의 의미는 해당 tuple의 개수를 나타낸다.

예시로 (1,a)는 해당 table에서 2개나 있기 때문에 λ=2이다.

Generalizing Set Operations to Multiset Operations

intersection

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

∩

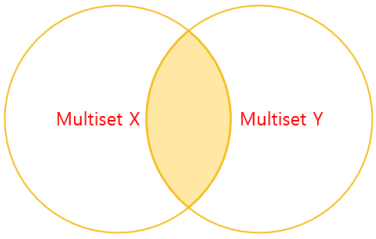
Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

=

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0



이렇게 **교집합**(intersection)은 서로 **공통으로 가지는 원소(tuple)의 집합**이고

DB에서는 intersection은 교집합 부분의 원소(tuple)들을 가지고 새로운 table을 만드는 것이다.

또 interection은 **오름차순으로 정렬되서 출력**함.

중복된 데이터를 출력한다.

여기서는 새로운 table이 Multiset Z이다.

식으로 표현하면 λ(Z) = min(λ(X), λ(Y))) 이다.

Union

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

∪

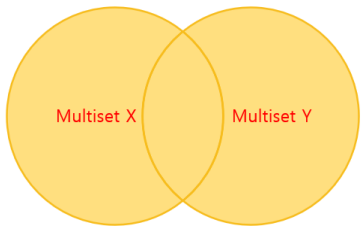
Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

=

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2



union은 합집합을 뜻하며, **각 집합의 원소들을 합하여 집합을 만든다.**

table끼리의 결합이라고 생각하면 된다.

또 union은 set집합이기 때문에 **중복을 허용하지 않고, 오름차순으로 정렬**해서 만들어줌!

식으로 표현하면 λ(Z) = λ(X) + λ(Y) 이다.

ex) A{1,3,5}, B(2,4,5) 일때 AUB = {1,2,3,4,5} 이다.

이해를 돕기위해 위의 예제를 이용해서 작성해보면

X = {(1, a), (1, a), (2, c), (2, c), (2, c)}

Y = {((1, a), (1, a),(1, a), (1, a), (1, a), (1, b), (2, c), (2, c), (1, d), (1, d)}

XUY = Z = {(1, a), (1, b), (2,c), (1, d)}가 원래 정답 이지만

위의 답은 다르다!

그 이유는 위에 예시는 단순히 union만을 쓴것이 아닌 중복값을 허용하는 union all 명령어를 사용했기 때문이다.!

이번 post처음에도 말했지만 집합연산자들은 중복을 허용하지 않는다.

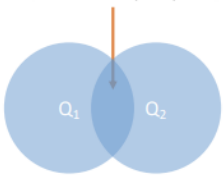
하지만 중복을 허용하는 방법은 있다.! union all과 같이

union all은 출력할때 정렬하지 않음!

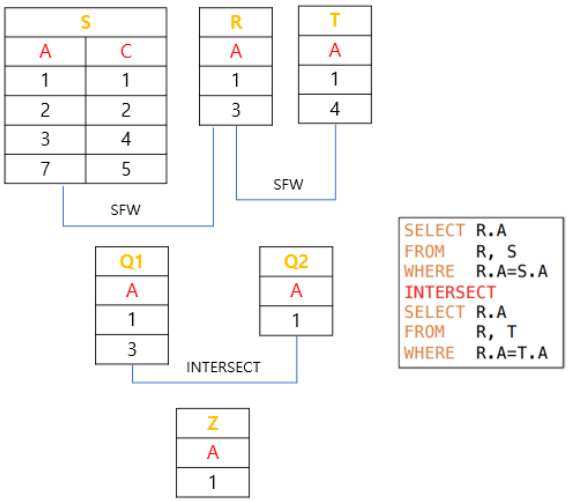
Explicit Set Operatiors : IMTERSECT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
INTERSECT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$



이 문제를 보면 INTERSECT를 명시하여 Q1,Q2 table의 교집합을 새로운 table로 만들려고 하는것을 알 수 있다.
INTERSECT기준 위 SFW는 Q1, 아래 SFW는 Q2이다.
풀이 과정 임의의 예시로 아래와 같다.



먼저 Q1, Q2 table을 만든다.
Q1을 만들기 위해 위의 query문을 보면
1. FROM절을 보면 R을 기준으로 잡고 S와 join하고 싶다는 의미
※위 방법은 암묵적인 방법이며, join할때 주의할 점으로 join할 때 두 table을 공통으로 묶어줄 attribute가 있어야됨!
2. WHERE절을 보면 R.A = S.A 의 value가 같은 tuple끼리 가로로 연결하라는 뜻이다.
※R.A와 같이 dot(.)이 들어간 이유는 두 table의 attribute의 이름이 동일하기 때문!
3. SELECT절을 보면 R.A 으로 R table의 A attribute를 column으로 해서 출력하라
이렇게 Q1이 만들어진다.

Q2도 윗 방법가 동일하게 만들어지며
4. INTERSECT절이 있어서 각 SFW로 임의로 만들어진 table의 교집합을 새로운 table로(여기선 Z) 하여 출력
이렇게 문장이 끝이 난다.

연산자로 집합만들때 조심할 점

첫번째 : 두 집합의 SELECT 절에 오는 column의 개수가 동일해야 한다.
두번째 : 두 집합의 SELECT 절에 오는 column의 데이터형이 동일해야 한다.
...

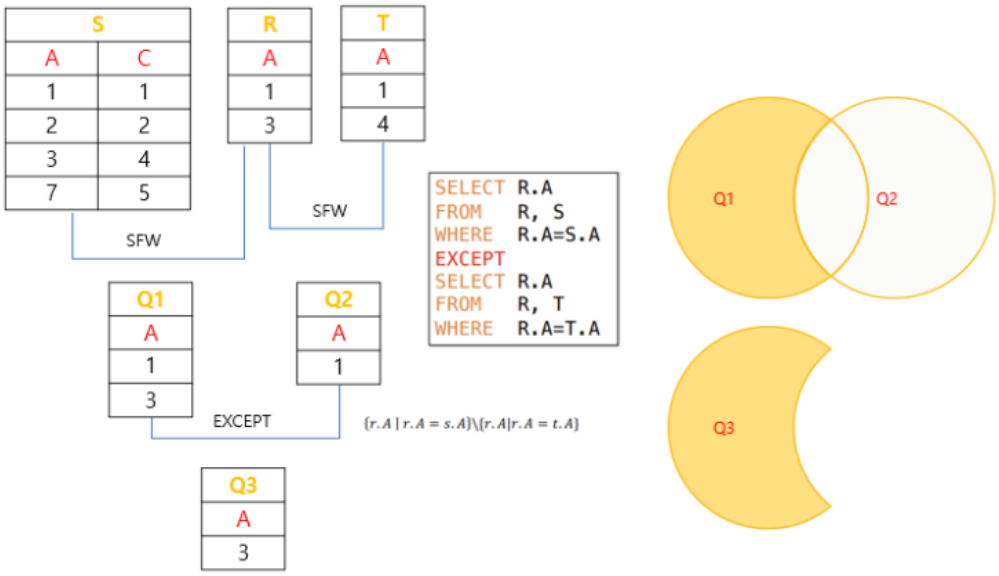
EXCEPT

EXCEPT operator는 **차집합은 만든때 사용**한다.

EXCEPT는

또한 중복을 허용하지 않는다.

중복은 허용하고 싶다면 다른 방법인 nested query를 사용해야 된다.

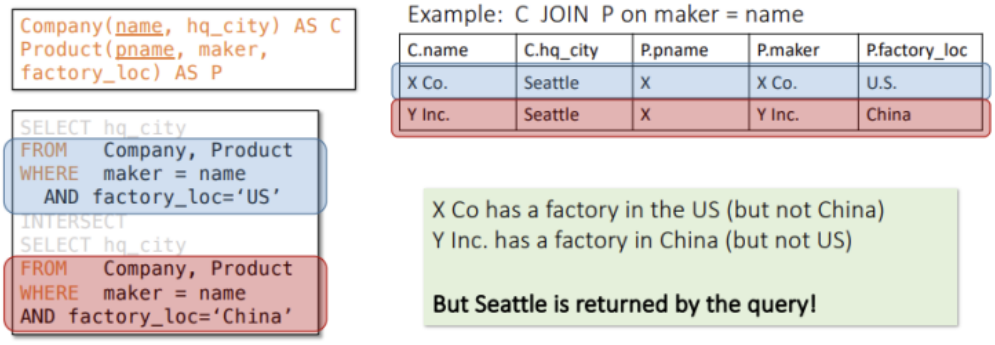


하나의 예시이며

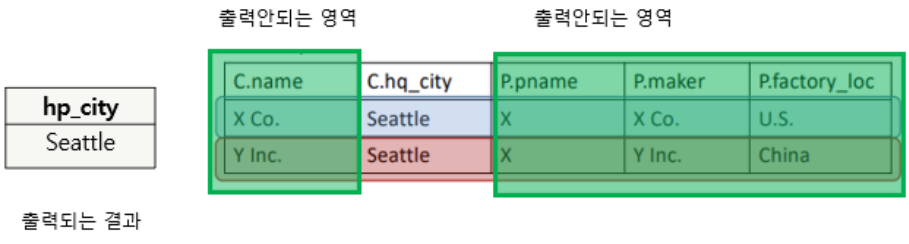
눈여겨 볼 점은 SELECT절의 attribute의 개수가 같으며,

데이터 타입도 동일하다는 것이다.

INTERSECT : Remember the semantics



문제 : Headquarters of companies which make gizmos in US AND China.



우리가 구하고 싶었던 부분은 "하나의 회사가 중국과 미국에서 Gizmos를 만들면서 hq_city가 한곳에 위치한 회사를 찾고 싶었는데"
출력된 결과는 Seattle 하나의 value를 가지는 2by1 matrix일것 이다.
왜냐하면 일단 중복을 허용하지 않기 때문에 하나의Seattle읖 value로 가진 table이 만들어 질것이고
이 결과는 factory in China and US를 만족하지 못한다.
이런 문제가 발생하기 때문에 nested query를 사용하면 된다.

One Solution : Nested Queries

- Nested Query는 중첩 질의라고 하며 SQL문 안에 또 다른 SQL문은 포함하는 구조이다.
- 그래서 중복된 값을 가질 수 있다.
- 그리고 sub query는 항상 ()로 묶어야 한다.
- 그리고 sub query는 단일 value or table을 반환할 수 있다.
- in() 안에서 또 다든 query를 실행을 하고 목록화해서 출력해줌.

```
Company(name, hq_city)
Product(pname, maker, factory_loc)
```

```
SELECT DISTINCT hq_city
FROM   Company, Product
WHERE  maker = name
       AND name IN (
            SELECT maker
            FROM   Product
            WHERE  factory_loc = 'US')
       AND name IN (
            SELECT maker
            FROM   Product
            WHERE  factory_loc = 'China')
```

문제 : Headquarters of companies which make gizmos in US and China

해석
Company를 기준으로 Product라는 table을 join하는데 그 조건은
maker와 name의 value가 같아야 되며,
sub query에서 Product table의 [factory_loc = US를 만족하는 tuple로](행) [maker attribute](열)이 있는 table을 반환하고 이 table의 maker의 value와 name의 value가 일치한다면 해당 tuple을 출력
그리고 Product table의 factory_loc = China를 만족하는 tuple 중 maker attribute의 value가 name의 value와 같다면 Company table의 hq_city attribute를 column으로 table을 만들어라.
※여기서 중요한 점은, IN이하 row와 name의 row가 하나라도 일치하면 그 row를 출력하고, 일치 하는지 비교할때는 하나의 row 씩 비교함

다든 문제로

Studnets			
ID	Name	Class_id	GPA
2201	Kang	3	3.45
2202	Choi	1	3.15
2203	Lee	1	4.30

2204	Park	2	3.10
2205	Kim	2	2.40

Teacher

ID	Name	Subject	Class_id	Monthly_salary
1	T_1	history	3	$2,500
2	T_2	literature	NULL	$2,000
3	T_3	english	1	$2,350
4	T_4	math	2	$3.000

Classes

ID	Grade(학년)	Teacher_id	Number_of_student
1	1	3	21
2	3	4	25
3	5	1	28

만약 평균 이상의 GPA를 가진 모든 학생을 찾고 싶다고 가정하면
우리가 평균 GPA 값을 모르기 때문에 찾아줘야되는 찾아주는 방법은

```
SELECT AVG(GPA)
FROM Students;
```

로 평균값을 구하는 방법도 있지만.
nested query로 한번에 가능도 하다.

```
SELECT *
FROM Students
WHERE GPA > (
    SELECT AVG(GPA)
    FROM Students);
```

이 query문을 해석해보면 Students table의 있는 모든 attribute를 포함하는 table을 만들고 싶은데 그 조건이 Students table의 GPA attribute의 values의 평균값 보다 큰 GPA를 가진 tuple을 필터링한 다음 만들어라.

ID	Name	Class_id	GPA
2201	Kang	3	3.45
2203	Lee	1	4.30

https://learnsql.com/blog/sql-nested-select/

nested query 참조했던 site 입니다

또 우리는 comparison operators or In, NOT IN, ANY or ALL 연산자들을 사용해서 where절에 nested query를 만들 수 있다.

[IN] operator는 **sub query에서 반환된 table에 특정 value가 있는 체크**한다.

[NOT IN] 연산자는 **sub query에서 반환된 해당 table에 없는 값에 해당하는 tuple(row)은 필터링** 한다.

[ANY] 연산자는 sub query에서 반환된 value가 이 조건을 충족 하는지 평가하기 위해 **비교 연산자와 함께 사용됨.**

[ALL] 연산자는 sub query에서 반환된 모든 value가 이 조건을 충족 하는지 평가하기 위해 **비교 연산자와 함께 사용됨.**

```
SELECT AVG(Number_of_students)
FROM Classes
WHERE Teacher_id IN (
    SELECT ID
    FROM Teachers
    WHERE Subject = 'English' or Subject = 'History');
```

문제를 보면 history or english를 가르치는 선생의 수업의 평균 학생 수를 계산해주는 query문이다.

이 qurey 문을 보자 여기서는 sub query에서 Techers table에서 Subject attribute의 value값으로 영어과목 or 역사과목을 가지고 있는 tuple중 ID attribute의 value를 반환하라는 뜻이다. (즉 ID column과 sub query condition을 충족하는 row들이 있는 table을 반환함.)
그 다음 Teacher_id의 모든 value에 대해 IN 연산자는 해당 ID가 sub query에서 반환된 table에 있는지 확인한다.
그리고 만약에 있으면 SELECT문을 출력하여 평균학생 수를 말해준다.
main query문에 sub query에 오는 table이 join이 되어야 된다고 생각했는데 잘못된 생각이였다.
join은 두 table을 합쳐서 컬럼의 갯수를 늘려 두테이블에 있는 여러정보를 비교 및 보고 싶을때 사용하는 것이기 때문에 필수는 아니다. 필요하면 쓰는 것이다.

Nested queries: Sub-queries Return Relations

sub query들은 relation들을 반환한다.
문제 : "Cities where one can find companies that manufacture products bought by Joe Blow"

```
Company(name, city)
Product(name, maker)
Purchase(id, product, buyer)
```

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
   AND p.buyer = 'Joe Blow')
```

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe Blow'
```

문제를 보면 sub query의 별칭 지정이 잘못되었다. product가 p이다. where절의 buyer도 그래서 pr로 바꿔야 된다.
푸는 방법은 두 개가 있고 오든쪽 방법은 3개의 table을 join하고 있기 때문에 성능면에서 안좋다.
왜냐하면 table size가 클수록 join의 성능이 안좋을 수 밖에 없기 때문이다.

왼쪽 query문을 해석해보면
1. Company table을 선택한다. 또 Company = c
2. **IN()안은 sub query**라고 하며 해석해보면 Purcase, Product table을 join할 것이고 공통으로 공유하는 attribute는 name 과 product이고 buyer = Joe Blow를 만족하는 tuple을 행으로하고 maker을 열로 하는 table을 만들어라.
3. c.name은 IN()절의 relation에 있는 value하고 같을때 city를 열로하는 table을 생성하라
※Join은 관련 있는 column을 기준으로 row를 합쳐주는 연산이다.
※ IN은 sub query에 먼저 접근하고, 그 다음 Company table의 row와 비교하여 일치하는 value가 있으면 해당 row를 출력함. , EXIST와 접근하는 순서가 반대다.

※주의할 점으로 이렇게 하면 데이터의 값에 따라 중복된 값이 출력될 수 있기 때문에 SELECT문의 DISTINCT를 사용해 준다.

Subqueries Return Relations

- ANY, ALL은 SQLite에서는 지원하지 않음.
- ANY는 sub query의 결과 중 한 줄씩 결과를 비교하며
- < ANY이면 sub query의 값보다 작은값들만 선택함.
- > ANY이면 sub query의 값보다 큰값들만 선택함.
- ALL은 sub query의 모든 결과를 비교하며 부등호는 수학의 부등호와 같은 역할을 한다.
- ALL과 ANY, IN의 반환 row는 여러개 일 수 있고, 반환 cloumn은 1개 이다.
- EXIST는 한개의 행과 열을 가져와서 비교, 반환 행은 여러개 일 수 있고, 반환 열도 여러개 일 수 있다.
  또 EXIST는 value가 존재하는지 확인할 때 사용하고, 존재하면 해당 value를 반환한다.
- IN은 한 개의 column을 가져와서 비교 반환 행은 여러개 이고, 반환 열은 한개 이다.
- NOT IN에서 주의할 점은 sub query의 결과중 NULL값이 있으면 NULL과의 비교operate는 항상 UNKNOWN값을 반환하여, WHERE이 FALSE가 되므로 해당 row가 출력되지 않는다.

prodcut

id	name
1	라면
NULL	사탕
2	자전거
3	오토바이
4	요트
5	NULL
6	NULL
7	조명
8	서랍
9	헬멧
NULL	마우스
NULL	키보드
10	모니터

color

id	color
1	black
2	green
3	blue
4	NULL
5	yellow
6	white
NULL	red

만약 위 처럼 두 개의 table이 있다고 가정을

IN

```
SELECT *
FROM product p
WHERE p.id IN (
    SELECT c.id
    FROM color c);
```

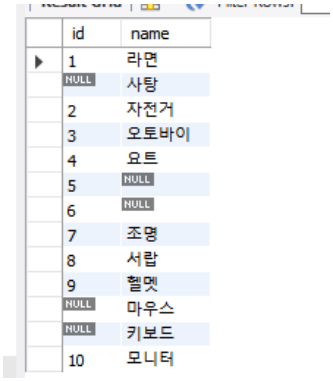
위와 같이 코드를 작성하게 되면 제일 먼저 sub query를 excute하고(즉, 소괄호 안의 sub query에 먼저 접근) 그에 대한 값을 먼저 가져오고 product에서 하나의 tuple씩 IN 이하의 sub query의 실행된 값들과 일치하는지 비교하고 IN 이하의 값들 중 하나라도 일치하면 그 tuple을 생성하게 됨.

id	name
1	라면
2	자전거
3	오토바이
4	요트
5	NULL
6	NULL

EXIST

```
SELECT *
FROM product p
WHERE EXISTS (
   SELET c.id
   FROM color c);
```

이대로 출력하게 되면 product table과 동일하게 출력되는 것을 확인 할 수있다.



id	name
1	라면
NULL	사탕
2	자전거
3	오토바이
4	요트
5	NULL
6	NULL
7	조명
8	서랍
9	헬멧
NULL	마우스
NULL	키보드
10	모니터

그 이유는 EXISTS는 IN과 달리 main query인 product에 먼저 접근해서 하나의 row를 가져오고 suq query를 실행하여 sub queyr의 대한 결과가 존재하는지 확인하고 존재하면 해당 row를 생성하기 때문이다.
여기서 예시로 1번째 row인 (1, 라면)을 먼저 가져왔고, sub query에 대해 어떠한 결과로도 존재하기만 한다면 True가 되어서 (1, 라면)이 결국 출력됨
그리고 해당 sub query는 항상 결과 값을 가지는 query이기 때문에 항상 참이 되어 모든 row가 실행되어 해당 table이 만들어진다. null도 값이다 unknown = 0.5
exists는p table의 row 하나당 c table의 모든 요소와 비교함.! 이때 c table의 null값이 있어 항상 참임.

만약 color와 product의 id가 같은 값이 있는 경우 table을 만들고 싶다면 아래와 같이 수정하면 된다.

```
SELECT *
FROM product p
WHERE EXISTS(
    SELECT c.id
    FROM color c
    WHERE p.id = c.id);
```

id	name
1	라면
2	자전거
3	오토바이
4	요트
5	NULL
6	NULL

NOT IN

```
SELECT *
FROM product p
WHERE p.id NOT IN (
    SELECT c.id
    FROM color c);
```

위와 같은 코드를 실행하면 아무런 결과도 나오지 않는다.
그 이유는 NOT IN이기 때문에 product의 tuple 값이 sub query의 결과 같이랑 같은 것이 없을때 출력하는데 sub query에 NULL값이 있기 때문에 아무런 결과도 안나온다.
왜냐하면 NULL과의 비교연산은 항상 unknown 값을 반환하므로 해당 조건문이 항상 FALSE가 되서 해당 tuple이 출력 안됨.

```
SELECT *
FROM product p
WHERE p.id NOT IN (
    SELECT c.id
    FROM color c
    WHERE c.id IS NOT NULL);
```

	id	name
▶	7	조명
	8	서랍
	9	헬멧
	10	모니터

위와 같이 sub query의 결과에 null 값이 없도록 만들면 처음에 원했던 결과를 얻을 수 있다.

NOT EXISTS

```
SELECT *
FROM product p
WHERE NOT EXIST (
    SELECT c.id
    FROM color c
    WHREE c.id = p.id);
```

위와 같이 실행하면 c.id = p.id가 참인 경우는 id가[1,2,3,4,5,6]인 경우 이고, product의 row가 이 요소를 가지고 있지 않으면 row를 출력하는데 product tuple의 NULL값이 비교값으로 들어갈때 NULL은 항상 unknown 값을 반환하므로 해당 row의 결과가 존재하지 않게 되어, 해당 row가 생성됨.

해당 내용 참조한 site 입니다.
https://doorbw.tistory.com/222

ANY, ALL operator 대소 비교

전체 데이터가 [1, 2, 3, 4, 5, 6, 7] 이고, sub-query 결과가 [4, 5] 일때

> ANY : sub-query의 **최솟값보다 큰 값은 찾아줌** -> 5, 6, 7

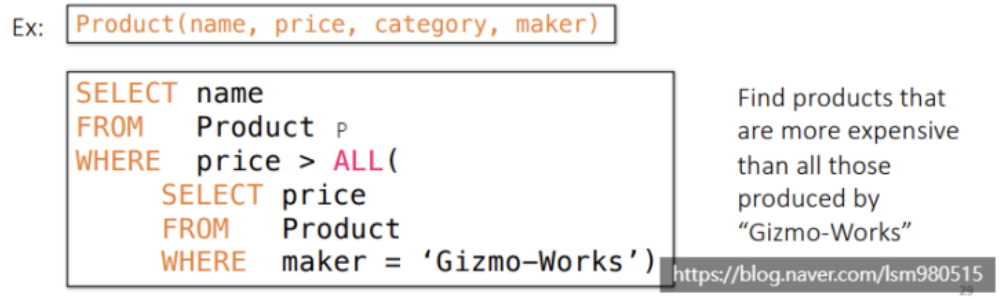
< ANY : sub-query의 **최댓값보다 작은 값 찾아줌** -> 1, 2, 3, 4

> ALL : sub-query의 결과 전부 or **최댓값보다 큰 값은 찾아 줌** -> 6,7

< ALL : sub-query의 결과 전부 or **최솟값보다 작은 값 찾아 줌** -> 1, 2, 3

https://doorbw.tistory.com/222
https://commontoday.tistory.com/190

IN,EXist...등을 참조한 site 입니다.



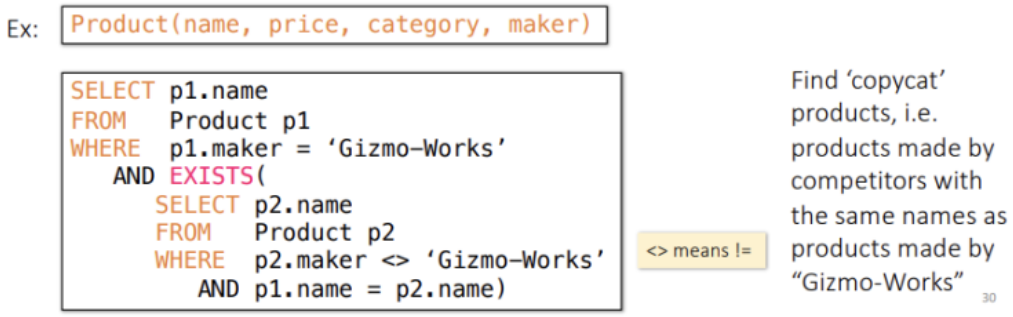
문제 : Gizomo_Works에서 만들어진 모든 상품들보다 비싼 제품을 찾고 싶다.

Product P

해석

1. Product table에서 maker가 Gizmo-Works 만족하는 경우 이 value를 값으로 가지고, field로 price를 가지는 table을 만들어라,

2. 이렇게 만들어진 table의 row와 Product table의 row중 price values의 값들을 비교해서 P.price의 가격이 더 높은 경우 name을 열로 가지는 table을 만들어라.



문제 : Gizmo-Works에서 만들어진 제품 인데, 경쟁 없체에서 만든 제품과 이름이 같은 'copycat'이라는 제품을 찾아라.

해석

1.Product(=p2) table에서 p2.maker의 value중 Gizmo-Works가 아닌 row를 찾고, p1.name과 p2.name이 같은 row가 있을시, p2.nama을 column으로 그리고 해당조건을 만족하는 value를 row로 하는 table을 만들어라

2. 그리고 Product(=p1) table에서 p1.maker의 value가 Gizmo-Works이고, sub query를 만족한 row가 있으면 p1.name을 column으로하고 해당 row들을 행으로하는 table을 생성하여라.

※EXIST는 sub query에 대한 결과가 존재하는지 확인하고 존재하면 Ture, 존재하지 않으면 False를 반환하고, sub query의 row에 먼저 접근하지 않고, Product p1 row에 먼저 접근한 다음 p1 row에 name value와 새롭게 만들어진 table의 p2.name의 value들과 하나씩 비교하면서 일치 하면 해당 row를 하나씩 출력하여 table을 만든다. (WHERE절의 참 거짓을 보고 읽은 행의 선택여부를 결정함.)

Correlated Queries

```
Movie(title, year, director, length)
```

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
          SELECT  year
          FROM    Movie
          WHERE   title =  m.title)
```

Find movies whose title appears more than once.

Note the scoping of the variables!

Note also: this can still be expressed as single SFW query...

상관쿼리라고 하며, 내부 sub query 안에 외부 변수를 사용하는 것이다.
즉 sub query 값이 결정될려면 Outer Query(외부 쿼리)의 값이 있어야 sub query가 수행됨.
AS는 ALISA라고 불리며, 별칭을 주는 개념으로 Movie를 다른이름으로 m으로도 나타낼 수 있게 해준다.
AS는 column, table, sub query, WHERE, FROM, SELECT절 등에서 사용될 수 있다.

해석
1. Movie table을 별칭으로 m이라고 하며,sub query의 Movie table에서 title과 m.title이 같을 때 해당 row를 생성하고 year을 attribute로 하는 table을 만든다. 그리고 year와 생성된 table의 row를 하나씩 비교해서 서로 다르면 해당 row를 선택
2. 출력된 row를 행으로하고 중복된 값을 포함하지 않는 title column을 열로하는 table을 만들어라.

```
Product(name, price, category, maker, year)
```

```
SELECT DISTINCT  x.name, x.maker
FROM   Product AS x
WHERE  x.price > ALL(
          SELECT y.price
          FROM   Product AS y
          WHERE  x.maker = y.maker
             AND y.year < 1972)
```

Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

can be very powerful(also much harder to optimize)

문제 : 1972년 이전에 같은 제조사에서 만들어진 제품들보다 더 비싼 제품을 찾아라.
해석
1. Product라는 table의 별칭을 x로하고(메인 쿼리), Product라는 table의 별칭을 y로 한다.(sub query)
2. sub query에서 x.maker = y.maker가 참이고, y.year < 1972가 참인 row들을 선택하고 이 row들을 행으로하고 y.price를 column으로 하는 table을 만들어라
3. 그리고 sub query의 값보다 x.price의 값이 크다면 해당 row들을 선택하고, x.name, x.maker를 중복값을 허용하지 않는 attribute를 column으로 하는 table을 만들어라.

Aggregation(집합)

Aggregation

```
SELECT AVG(price)
FROM   Product
WHERE  maker = "Toyota"
```

```
SELECT COUNT(*)
FROM   Product
WHERE  year > 1995
```

- SQL supports several **aggregation** operations:
  - SUM, COUNT, MIN, MAX, AVG

집합 연산자(집계 함수)들은 [SUM(총합), COUNT, MIN(최소), MAX(최대), AVG, STDDEV(표준편차), VARIANCE(분산)]이다.
집계 함수는 null 속성 값은 제외하고 계산하며, WHERE절에는 집계 함수 사용x, HAViNG절에서 사용가능
COUNT는 출력이 된 record의 개수 이며, 보통 where절과 같이 사용 됨.
where절이 없다면 해당 table의 모든 tuple의 수를 계산한다.

- COUNT applies to duplicates, unless otherwise stated

```
SELECT COUNT(category)
FROM   Product
WHERE  year > 1995
```

Note: Same as COUNT(*). Why?

We probably want:

```
SELECT COUNT(DISTINCT category)
FROM   Product
WHERE  year > 1995
```

count() <- ()에는 column이 오고 column이 오는 경우 NULL은 포함하지 않고 count하고, *는 NULL을 포함해서 count한다.
즉 *가 오면 where절 조건을 만족하는 모든 record를 count한다고 생각하면 된다.

```
Purchase(product, date, price, quantity)
```

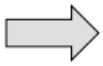
```
SELECT SUM(price * quantity)
FROM   Purchase
```

SUM은 총합을 나타내 주므로, SUM(가격*갯수)은 총매출을 출력하게 된다.

Purchase

Product	Date	Price	Quantity
bagel	10/21	1	20
banana	10/3	0.5	10
banana	10/10	1	10
bagel	10/25	1.50	20

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = 'bagel'
```



50 (= 1*20 + 1.50*20)

이렇게 되면 where절이 있기 때문에 bagel이 있는 row의 price와 quntity의 곱의 합을 출력하게 된다.
null값이 table에 존재하더라도 해당 null값은 계산할때 0으로 취급함.

Grouping and Aggregation

GROUP BY절은 **해당 column값중 동일한 값은 그룹화**해서 해당 record를 묶어서 자료를 관리할 때 사용되고,

- 그둡함수와 함께 사용을 많이 한다.
- 또 GROUP BY를 사용할 때 WHERE절은 조심해서 사용해야 된다.

왜냐하면 WHERE절이 GROUP BY보다 내부적으로 먼저 처리 되기 때문이다.

그래서 GROUP BY에서 조건을 주고 싶으면 HAVING절을 사용한다.

- GROUP BY를 사용하면 해당 column은 unique하므로 distinct를 사용할 필요가 없다.
- 또 GROPU BY에 지정한 column 이외의 column은 집계함수(aggregate function)를 사용하지 않은 채 SELECT문을 기술해서는 안됨.
- 즉 name을 GROPU BY했다면 name이외의 column을 SELECT문에 적을때 집계함수를 사용해야됨.
- **DB처리 순서는 아래와 같다.**

FROM-> WHERE -> GROUP BY -> HAVING -> SLECT -> ORDER BY

Purchase

product	date	price	quantity
bagel	10/25/2005	1	20
banana	10/3/2005	0.5	10
banana	10/10/2005	1	10
bagel	10/25/2004	1.5	20

```
SELECT   product,
         SUM(price * quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Find total sales after 10/1/2005 per product.

Purchase

product	date	price	quantity
bagel	10/25/2005	1	20
	10/25/2004	1.5	20
banana	10/10/2005	1	10
	10/3/2005	0.5	10

오든쪽과 같은 문제는 위와 같은 명령어로 나타낼 수 있다.

해석

1. Puchase table에서 2005년 10월 1일 이후 값을 가진 row들 중 product의 value을 그둡화 하여라
2. 그리고 product, SUM을 column으로 하는 table을 만들어라.
3. 단 SUM의 별칭은 TotalSales이고, sum의 각 row의 value는 price*quantity이다.

※ 만약 as가 없었다면 sum column명이 SUM(price * quntity)로 출력되었음

※DB처리 순서는 아래와 같다.

FROM-> WHERE -> GROUP BY -> HAVING -> SLECT -> ORDER BY

product	TotalSales
bagel	20
banana	15

HAVING Clause

HAVING절은 GROUP BY절 다음에 오는 조건절임.

HAVING절은 GROUP BY절의 결과에 조건을 줄때 사용함.(실행순서가 GROUP BY가 먼저이기 때문에 결과에 조건을 줌.)

WHERE을 GROUP BY의 조건절로 사용할 수 없다.

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING   SUM(quantity) > 100
```

GROUP BY절까지 진행했다고 가정하고 Having절을 보면 SUM(quantity)의 의미는 bagel의 sum(quantity)는 20이고, banana의 sum(quantity)도 20이다 따라서 HAVING절을 만족하지 않으므로 해당 쿼리문은 실행되어도 테이블의 값은 없다.

왼쪽은 nested query방식으로 푸는 방법

오른쪽은 group by 방식으로 푸는 방법이다.

Group by와 관련되서 참조한 site 입니다.

https://gunso5.tistory.com/1133

Group by와 관련되서 참조한 site 입니다

Group by vs. Nested Query

SFW가 한 번 등장하는 group by가 더 효율적인 방법이다.

Quantifiers

Quantifier(정량자)는 universal quantifier(전칭 기호: ∀)와 existential quantifier(존재 기호: ∃)가 있다.
모든 x에 대하여 명제 F(x)가 성립할 때 $\forall xF(x)$ 로 나타내고, 명제 F(x)가 성립하는 x가 존재할 때 $\exists xF(x)$로 나타낸다.
관계 대수는 원하는 정보와 그 정보를 검색하기 위해 어떻게 유도하는 가를 기술하는 '절차적 언어'이다.
관계 해석은 "무슨 결과를 얻고 싶은가"의 관점이다. 그래서 단지 원하는 결과가 무엇인지 알아내는 것이므로 '비절차적 언어'라고 불림
※비절차적 언어는 무엇인지만을 명시하고 query를 어떻게 수행할 것인가는 명시하지 않는 것을 의미
관계 대수로 표현한 식은 관계해석으로 표현할 수 있고, tuple 관계해석과, domain 관계해석이 있다.
tuple 관계해석은 전체 table 중에서 얻고 싶은 결과의 record들이 어떤 것들인지 규정한 것.
- {T | p(T))의 형식을 가짐
T =t 은 튜플 변수이며, p()는 T를 기술하는 공식임.
따라서 이 질의의 결과는 공식 p(T)가 T=t의 값으로 참(True)이 되는 모든 투플 t의 집합이다.
이러한 공식 p(T)를 작성하는 언어는 TRC의 핵심이고 기본적으로 일차 논리의 부분집합이다.
튜플 변수는 어느 특정 relation의 튜플들을 값으로 취하는 변수.
즉, 주어진 튜플 변수에 대입되는 값들은 동일한 개수와 타입의 filed들을 가짐.
변수의 타입이 일치하지 않는 값들의 비교는 항상 실패한다.
하나의 공식에 어떤 변수가 있을때 그 공식이 그 변수를 속박하는 어떤 정량자를 포함하지 않으면, 그 변수는 자유돕다라고 말함. ex) ∃R(p(R)), 이때 ∃이 변수 R을 속박한다라고 함.
※아래의 표는 관계해석은 연산자 표이다.

연산자	or 연산	∨	원자식 간 '또는' 관계로 연결
	and 연산	∧	원자식 간 '그리고' 관계로 연결
	not 연산	¬	원자식에 대해 부정

domain 관계해석은 구하고자 하는 결과의 특정 attribute의 범위를 따로 규정해 놓은 것,

튜플 관계해석

Sailors			
sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0

74	Horatio	9	35.0

질의 : 등급이 7보다 큰 모든 뱃사람을 구하시오
답 : {S | S ∈ Sailors ∧ S.rating > 7 }
해석
1. S는 Sailors의 원소이면서, Sailors table의 rating이 7보다 크면 S는 Sailors의 원소이다.
그 결과 sid 값 31, 32, 58, 71, 74를 가진 Sailors의 tuple로 구성됨.

TRC 질의의 의미

TRC 질의는 {T | p(T)}의 형식으로 정의되며, 이때 T는 공식 p내에서 유일한 자유 변수이다.
TRC 질의에 대한 답은 변수 T에 튜플값 t를 대입하여 식 p(T)가 참이 되게 하는 튜플의 집합이다.
어떤 식에 있는 자유 변수들에게 어떤 tuple값들을 대입하면 그 식이 참이 되는지를 명시하여야 한다.

질의: 등급이 7보다 큰 뱃사람들의 이름과 나이를 구하시오
답 : {P | ∃S ∈ Sailors(S.rating > 7 ∧ P.name = S.sname ∧P.age = S.age)}
해석
1. ()부분은 P를 기술하는 공식 부분이다.
2. P는 공식 Sailors() 내에서 유일한 자유변수 이다.
P는 name과 age의 filed를 가지고 있는 투플 변수로 간주된다.
왜냐하면 이 filed들이 여기에서 언급된 P의 유일한 filed들이기 때문이다.
P는 이 질의에 나오는 relation들의 어느 tuple도 취하지 않는다.
※ 잘모르겠지만 이 뜻이 의미는 P ∈ Relname으로 아직 P의 tuple에는 아무 값도 없다.
3. 이 질의의 결과는 두 filed name과 age를 가진 relation이 된다.
4. 원자식 P.name = S.sname과 P.age = S.age는 하나의 결과 튜플 P의 필드에 해당 값을 할당함.
그 결과 <Lubber, 55.5>, <Andy, 25.5>, <Rusty, 35.0>, <Zorba, 16.0>, <Horatio, 35.0> 튜플들의 집합이다.

관계해석의 정량자로 아래의 두개가 존재함.
- ∀는 'for all'로 읽음, 모든 가능한 tuple
- ∃는 'there exist'로 읽음, 어떤 tuple 하나라도 존재의 유무만 관심 있음

구분	관계대수	관계해석
특징	절차적 언어	비절차적 언어
목적	어떻게 유도하는가	무엇을 얻을 것인가
종류	순수관계 연산자(select, project, join, division), 일반집합 연산자(union, intersection, difference(minus), cartesian product)	tuple 관계 해석, domain 관계 해석

```
Product(name, price, company)
Company(name, city)
```

Find all companies with products all having price < 100

```
SELECT DISTINCT Company.cname
FROM   Company
WHERE  Company.name NOT IN(
     SELECT Product.company
     FROM Product.price >= 100)
```

Equivalent

Find all companies that make only products with price < 100

문제가 all의 가진 형식이라면 ∀(universal quantifier)이다.
이걸 only로 바꿔 줄려면 ∃을 사용하고 not in이 있기 때문에 ㄱ도 사용해야 된다.

해당 정보는 Database mangement systems 3rd edition을 참조해서 만들었습니다.

NULLS in SQL

- NULL은 아직 미정이라는 상태로 값이 0인 것과는 다르다.
- NULL값은 UNKNOWN이다. (즉, 미정)
- 비교 연산자로 두개의 NULL 값을 비교하면 항상 UNKNOWN 이다.
- 수식 연산에서 null값이 있으면 해당 결과 값은 null값이다.
- boolean operation에서 null = unknown이고 0.5의 값을 가진다.

```
SELECT *
FROM   Person
WHERE (age < 25)
  AND (height > 6 AND weight > 190)
```

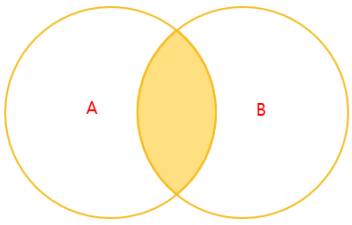
Won't return e.g.
(age=20
height=NULL
weight=200)!

이 질의문에서 입력되는 값중 null값이 있는 tuple을 출력되지 않는다.

```
SELECT *
FROM   Person
WHERE  age < 25 OR age >= 25
    OR age IS NULL
```

이 질의문에서 or age is null이 없으면 age에 null값이 있는 사람은 출력이 안되는데
or age is null을 입력함으로써 null값이 있는 사람도 출력이 됨.

Inner joins



```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
  JOIN Purchase ON Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

Both equivalent:
Both INNER JOINS!

명시적인 방법은 위에 방법과 같고 아래 방법이 암묵적 방법이다.
inner join은 두 table의 공통의 데이터에 대허서만 table을 합쳐서 사용하는 것이다. (교집합과 같다.)
JOIN 앞에 INNER을 적어도 되고 생략해도 된다.
만약 두 table중 기준 filed의 값 중 null값을 가지면 해당 tuple의 값은 출력하지 않음.

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

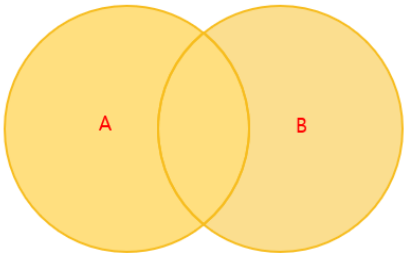
```
SELECT Product.name, Purchase.store
FROM   Product
  INNER JOIN Purchase
      ON Product.name = Purchase.prodName
```

Note: another equivalent way to write an INNER JOIN!

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

이 질의문을 해석해보면
Product table과 Purchase를 inner join하는데 공통으로 공유하는 attribute는 Product.name과 Purchase.prodName이고 이 값이 같으면 해당 tuple을 정보를 가지고 있고.
name, store를 coloum으로 하는 table을 출력하라.

Outer Joins



Outer join은 기준 filed 값이 어느 한쪽에만 존재해도 해당 tuple을 가져옴. (합집합과 같다)
그리고 기준 filed에 null값이 있어도 가져온다.
즉 조건을 만족하지 않아도 기준이 되는 table에 해당하는 데이터는 모두 보여준다.

Left outer join

- Join 기준 왼쪽 table에 있는 모든 record를 가져오고, 기준 filed의 값과 매칭이 되는 레코드들도 가져온다.
- 특징으로 왼쪽 table filed의 기준 filed 값과 매치되는 값이 오든쪽 table에 없으면 오든쪽 table의 filed에 null 값을 출력
- OUTER 생략가능

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM   Product
  LEFT OUTER JOIN Purchase
       ON Product.name = Purchase.prodName
```

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

위 질의문을 해석해보면
Product를 기준으로 Purchase table을 합치라는 뜻인데
기준은 Product.naem = Purchase.prodName의 값을 비교해서 값이 일치하는 row끼리 가로방향으로 연결하라는 뜻이다.
그리고 연결안된 row들 중 왼쪽 table에는 값이 있지만, 오든쪽 table에는 값이 없을시 null로 출력한다.
그리고 join해서 생긴 새로운 테이블에서 product table의 name column과 purchase table의 store column을 출력하라는 뜻이다.

단 왼쪽 table에는 값이 있지만, 오든쪽 table에는 값이 없을시 null로 출력한다.
그리고 name, store을 column으로 하는 table을 만든다.

Full outer join

Full outer join은 left join과 right join을 합친 것이다.
즉, 양쪽 모두 조건이 일치하지 않는 것들까지 모두 결합하여 출력함.
공통으로 공유하는 coloum의 값이 둘다 null일때도 출력하고 싶을때 사용함.

https://jhkang-tech.tistory.com/55

Join과 관련되서 참조한 사이트 입니다.

Prof. Lee Seungjin님이 제공해준 Data Processing Systems의 자료를 활용하여 제작하였습니다.