효율적 쿼리

전제윤

서브쿼리

내부에 다른 SQL문이 포함되어 있는 SQL문

다른 SQL문 안에 있는 SELECT문(MySQL 공식문서)

서브쿼리

SELECT문 안에 SELECT문이 포함된 서브쿼리

외부쿼리

```
SELECT
    user.username,
    (SELECT COUNT(*)
    FROM posts
    WHERE posts.user_id = user.user_id) AS post_count
FROM user;
```

서브쿼리

username	post_count
Kim	2
lee	1
park	1

서브쿼리

DELETE문 안에 SELECT문이 포함된 서브쿼리

외부쿼리

```
DELETE FROM posts
   WHERE user_id = (
      SELECT user_id
      FROM users
      WHERE email = ‘kim@example.com'
);
```

서브쿼리

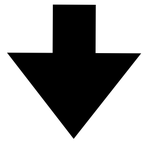
조인

여러테이블을 하나로 합치는 것

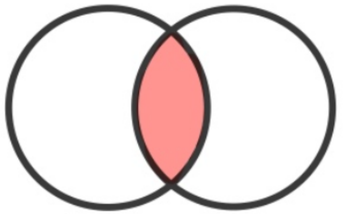
조인 INNER 조인

테이블 A와 B의 레코드 중 조인 조건을 모두 만족하는 레코드를 결과로 반환한다.

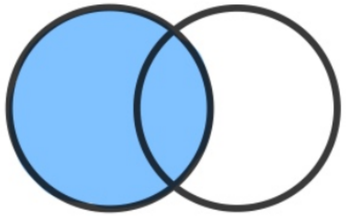
```
SELECT 필드
FROM 테이블1
    INNER JOIN 테이블2 ON 조인 조건;
```



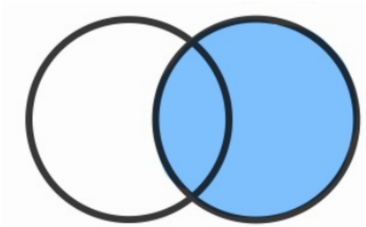
```
SELECT customers.name, customers.age
FROM customers
    INNER JOIN orders ON customers.id = orders.customer_id;
```



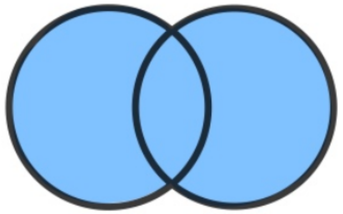
조인 OUTER 조인



LEFT OUTER 조인



RIGHT OUTER 조인

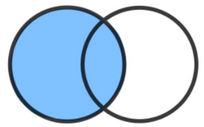


FULL OUTER 조인

조인 LEFT OUTER 조인

테이블1의 모든 레코드를 기준으로 테이블2의 레코드를 합치되,
테이블2에 대응되는 레코드가 없다면 해당 값을 NULL로 간주하는 조인방식

```
SELECT 필드
FROM 테이블1
    LEFT OUTER JOIN 테이블2 ON 조인 조건;
```



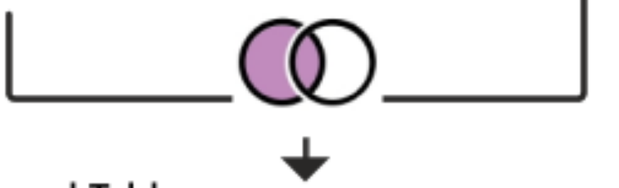
조인 LEFT OUTER 조인

Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

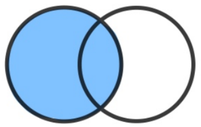
Right Table

ID	Country
1	USA
2	Canada
3	Panama



Merged Table

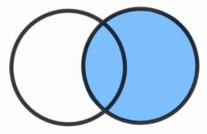
Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	*null*



조인 LIGHT OUTER 조인

테이블2의 모든 레코드를 기준으로 테이블1의 레코드를 합치되,
테이블1에 대응되는 레코드가 없다면 해당 값을 NULL로 간주하는 조인방식

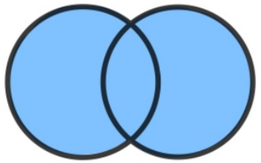
```
SELECT 필드
FROM 테이블1
    LIGHT OUTER JOIN 테이블2 ON 조인 조건;
```



조인 FULL OUTER 조인

두 테이블의 모든 레코드르 선택하되,
대응되지 않는 모든 레코드를 NULL로 표기하는 조인 방식.

```
SELECT 필드
FROM 테이블1
    LEFT JOIN 테이블2 ON 조인 조건;
    UNION
SELECT 필드
FROM 테이블1
    LIGHT JOIN 테이블2 ON 조인 조건;
```



조인 FULL OUTER 조인

[예제 테이블]

TABLE : FOOD_A	
ID	FOODNM
1	돈까스
2	삼겹살
3	초밥
4	곱창전골

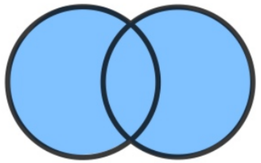
TABLE : FOOD_B	
ID	FOODNM
1	초밥
2	돈까스
3	칼국수
4	햄버거

[SQL예시]

```
SELECT *
FROM FOOD_A A FULL OUTER JOIN FOOD_B B
ON A.FOODNM = B.FOODNM;
```

[결과]

A.ID	A.FOODNM	B.ID	B.FOODNM
1	돈까스	2	돈까스
2	삼겹살	(NULL)	(NULL)
3	초밥	1	초밥
4	곱창전골	(NULL)	(NULL)
(NULL)	(NULL)	3	칼국수
(NULL)	(NULL)	4	햄버거



뷰

SELECT문의 결과로 만들어진 가상의 테이블

```
SELECT user.username, users.email, posts.title
  FROM users, posts
  WHERE users.user_id = posts.user_id ;
```

username	Email	Title
Kim Kim Lee Park	kim@example.com kim@example.com lee@example.com park@example.com	One Two Three Four

뷰

SELECT문의 결과로 만들어진 가상의 테이블

```
SELECT result.username, result.email, reuslt.title
   FROM
   (SELECT user.username, users.email, posts.title
   FROM users, posts
   WHERE users.user_id = posts.user_id ) AS result
   WHERE result.username = ‘kim’
```

result.username	reuslt.Email	result.Title
Kim Kim	kim@example.com kim@example.com	One Two

뷰

SELECT문의 결과로 만들어진 가상의 테이블

```
CREATE VIEW myview AS
  SELECT user.username, users.email, posts.title
  FROM users, posts
  WHERE users.user_id = posts.user_id ;
```

myview

username	Email	Title
Kim Kim Lee Park	kim@example.com kim@example.com lee@example.com park@example.com	One Two Three Four

```
SELECT result.username, result.email, reuslt.title
  FROM
  (SELECT user.username, users.email, posts.title
  FROM users, posts
  WHERE users.user_id = posts.user_id ) AS result
  WHERE result.username = ‘kim’
```

```
SELECT username, email, title
  FROM myview
  WHERE username = ‘kim’
```

result.username	reuslt.Email	result.Title
Kim Kim	kim@example.com kim@example.com	One Two

인덱스

검색 속도 향상을 목적으로 만드는 하나 이상의 테이블 필드에 대한 자료구조.

1. 클러스터형 인덱스

2. 세컨더리 인덱스

0

클러스터형 인덱스

- 생성시에 데이터 페이지 전체가 다시 정렬되므로,
  서비스가 운영중에 클러스터형 인덱스를 생성하면 큰 부하
- 보조 인덱스보다 검색 속도가 빠르고, 변경(INSERT, UPDATE, DELETE)는 느림
- 평균적으로 보조 인덱스보다 훨씬 빠르지만, 테이블에 한 개만 생성할 수 있음

인덱스

보조 인덱스

- 생성시에 데이터 페이지는 건드리지 않고, 별도의 페이지에서 인덱스를 구성하는 작업을 실행
- 클러스터형 인덱스보다 검색 속도는 느리고, 변경(INSERT, UPDATE, DELETE)은 덜 느림
- 보조 인덱스는 여러 개 생성할 수 있지만, 충분히 고려하고 사용해야 함

감사합니다.