8253jang@daum.net
장희정

JOIN

한번의 SELECT 문장으로 2 개 이상의 테이블에 있는 컬럼의 정보를 검색하고 싶을 때 사용한다.

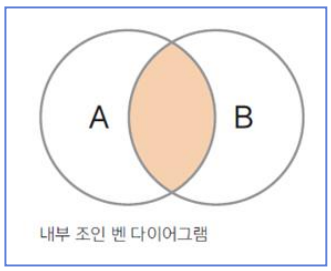
JOIN 의 종류

1) INNER JOIN

- EQUI JOIN = 동등조인 = NATURAL JOIN
- NON EQUI JOIN : 조인 대상 테이블의 어떤 컬럼의 값도 일치하지 않을 때 사용
  EX) BETWEEN AND , IS NULL, IS NOT NULL, IN, > , < 등의 조건문을 사용할 때 쓴다

보통 조인이라고 하면 내부 조인(INNER JOIN)을 의미할 정도로 가장 많이 쓰며 두 테이블 모두 조건에 맞은 열을 조회할 수 있다

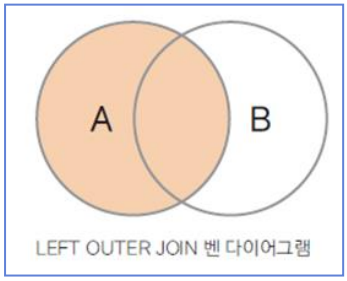
다이어그램으로 표현하면 다음과 같다.

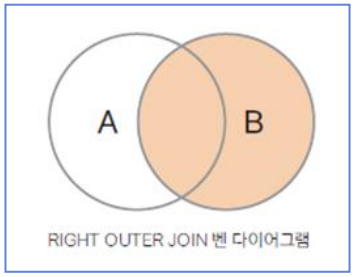


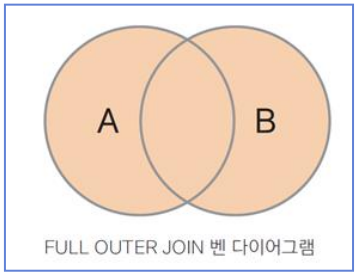
2) OUTER JOIN

기본 EQUI JOIN 을 하면서 별도의 테이블의 모든 정보를 검색하고 싶을 때 사용한다.

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN (MySQL 은 full 지원안함 : left, right 조인 후 union 해서 full 조인 효과)







3) SELF JOIN

자기 자신테이블을 조인하는 것(하나의 테이블을 2 개처럼 사용하는 것)

주로 재귀적 관계일 때 많이 사용한다.

(재귀적관계란 하나의 테이블에서 일반속성이 자기자신테이블의 PK 를 FK 로 참조하는 것)

8253jang@daum.net
장희정

JOIN 코딩 방법

1) SQL JOIN - 암시적 조인(Implicit Join)
2) ANSI JOIN – 명시적 조인(Explicit Join)
미국국립표준협회(American National Standards Institute, ANSI)에서 정한 미국의 표준을 기본으로 한다. – 권장

암시적 조인(Implicit Join)

Ex) SELECT * FROM TEST1 , TEST2 WHERE TEST1.ID=TEST2.ID;

TEST1, TEST2 를 사용하여 **데카르트 곱(Cartesian Product)** 을 먼저 생성한 후 WHERE TEST1.ID=TEST2.ID 조건을 적용해서 원하는 결과를 얻는다.

내부적으로 MySQL 은 CROSS JOIN 후 WHERE 필터링을 수행하는 방식과 동일하게 처리한다.

예를 들어, TEST1 에 3 개의 행, TEST2 에 4 개의 행이 있다면, **12 개의 조합이 먼저 생성된 후** WHERE TEST1.ID=TEST2.ID 조건을 만족하는 행만 필터링된다.

이 방식은 과거의 SQL 표준 방식이었지만, 현재는 명시적 JOIN 을 사용하는 것이 일반적이다.

명시적 조인(Explicit Join)

Ex) SELECT * FROM TEST1 JOIN TEST2 ON TEST1.ID = TEST2.ID;

JOIN ... ON 절을 사용하여 두 테이블을 조인한다.

MySQL 의 **최적화 엔진이 JOIN 조건을 미리 고려**하여 불필요한 데카르트 곱을 만들지 않고, 바로 조건에 맞는 행들만 결합한다.

따라서 명시적 JOIN 을 사용하면 **읽기 쉬운 코드가 되고, 성능 최적화도 더 효율적**일 가능성이 높다.

SQL INNER JOIN 형식

```
SELECT [열]
FROM [테이블 1] , [테이블 2]
WHERE [조인조건]
```

ANSI INNER JOIN 형식

```
SELECT  [열]
FROM  [테이블 1]
INNER JOIN [테이블2] ON [테이블1.열] = [테이블2.열]
WHERE [검색 조건]
```

8253jang@daum.net
장희정

ANSI OUTER JOIN 형식

```
SELECT  [열]
FROM [테이블1]
        [LEFT | RIGHT | FULL] OUTER JOIN [테이블 2] ON [테이블 1.열] = [테이블 2.열]
WHERE [검색 조건]
```

USING 을 사용한 JOIN

JOIN 할 때 특정 컬럼(공통 컬럼)을 명확하게 지정하는 방법이다.

ON 은 조인할 때 조건을 직접 지정하지만 USING 은 공통 컬럼이름을 한번만 써도 자동으로 JOIN 된다.

```
SELECT ID, NAME, JOB, SAL
FROM TEST1 JOIN TEST2
USING(ID);
```

NATURAL JOIN

NATURAL JOIN 은 **동일한 이름을 가진 컬럼을 자동으로 기준으로 삼아 JOIN 을 수행**하는 방식

Ex) SELECT * FROM TEST1 **NATURAL JOIN** TEST2;

TEST1 과 TEST2 에서 같은 이름을 가진 컬럼(공통 컬럼)을 자동으로 찾는다.
그 컬럼을 기준으로 INNER JOIN 수행한다.
공통 컬럼은 결과에 한 번만 나타난다. (중복 제거)

SELF JOIN

같은 테이블을 두 번 사용하여 조인하는 방식으로 하나의 테이블을 마치 두 개의 서로 다른 테이블처럼 간주하고, 특정 컬럼을 기준으로 자신과 조인하는 것이다.
하나의 테이블을 **A, B 처럼 두 개의 별칭(Alias)으로 구분**하여 JOIN 을 수행한다.
EX) 계층적 관계(부모-자식 , 상사 – 직원) ,
짝을 이루는 데이터 매칭(멘토-멘티 , 소셜네트워크 친구관계)

```
SELECT A.컬럼명, B.컬럼명
FROM 테이블명 A JOIN 테이블명 B ON A.조인_조건 = B.조인_조건;
```

8253jang@daum.net
장희정

UNION 연산자

여러 쿼리의 결과를 하나의 데이터 집합으로 결합하는데 사용하는 명령문이다.
여러 개의 SQL 문의 결과에 대한 합집합으로 결과에서 모든 중복된 행은 하 나의 행으로 만든다.

UNION ALL 연산자

여러 개의 SQL 문의 결과에 대한 합집합으로 중복된 행도 그대로 결과로 표시한다

SUBQUERY - 부질의

- 서브 쿼리(subqueries)는 쿼리 안에 포함되어 있는 또 다른 쿼리를 말한다.
- ()괄호로 묶으며 괄호 안에 쿼리가 먼저 실행된 후 그 결과를 메인 쿼리의 조건으로 주로 사용한다.
- 서브쿼리의 **결과 행이 한 개일** 때 **비교연산자** 사용한다.
- 서브쿼리의 **결과 행이 여러 개 일** 때는 **ANY, ALL, IN** 연산자를 사용한다.
- 주로 SELECT 에서 많이 사용하지만 CREATE, INSERT, UPDATE ,DELTE, HAVING, WHERE , FROM ,ORDER 에서도 사용가능 하다.

Ex) EMP 사원테이블에서 평균 급여보다 더 많이 받는 사원 검색하고 싶다.

1) 평균 급여를 구한다.

```
SELECT AVG(SAL) FROM EMP;
```

2) 1)에서 구한 평균급여를 조건으로 사용한다.

```
SELECT * FROM EMP   WHERE SAL > 평균급여;
```

위, 1) 2)를 SUBQUERY 를 이용하여 하나의 문장 만든다.

```
SELECT * FROM EMP
WHERE SAL > (SELECT AVG(SAL) FROM EMP );
```

SUBQUERY 에 ANY 와 ALL 연산자 사용

ANY 는 **서브쿼리의 결과 중 하나라도 조건을 만족하면 참(TRUE)이 되는 연산자**
ALL 은 **서브쿼리의 결과값 중 모든 값과 비교하여 조건을 만족해야 하는 연산자**

-ANY

WHERE 컬럼 비교연산자 ANY (서브쿼리)

8253jang@daum.net
장희정

- ✓ 서브쿼리에서 반환된 값 중 **하나라도** 비교 조건을 만족하면 TRUE 이다
- ✓ ANY 는 IN 과 유사하지만, 비교 연산자(>, <, =, !=, >=, <=)를 사용할 수 있다는 점이 다르다

ex) 컬럼명 < any(100, 200, 300) => 최대값보다 작다
컬럼명 > any(100, 200, 300) => 최소값 보다 크다

-ALL

EX) WHERE 컬럼 비교연산자 ALL (서브쿼리)

- ✓ 서브쿼리에서 반환된 **모든 값과 비교했을 때 참(TRUE)이면 결과를 반환**한다.
- ✓ 예를 들어 SALARY > ALL (서브쿼리)이면, **서브쿼리의 가장 큰 값보다 커야 한다.**.
- ✓ ALL 은 최솟값, 최댓값과 조합하여 사용할 때 유용하다.

ex) 컬럼명 < all(100, 200, 300) => 최소값보다 작다
컬럼명 > all(100, 200, 300) => 최대값보다 크다.