[SQL 활용] 서브쿼리

2017.12.06 00:19

#All #ALL 비교연산자 #Any #ANY 비교연산자 #correlated #DBMS #exists #having 서브쿼리 #In #insert 서브쿼리 #MSSQL #MYSQL #NOT EXISTS #not in #Oracle #SQL #sql 가이드 #sql 서브쿼리 #sql 활용 #subquery #SXISTS #Un-Correlated #update 서브쿼리 #View #view DML #view 삭제 #VIEW 생성 #비교연산자 #서브쿼리 ANY ALL #서브쿼리란 #서브쿼리특징 #서브퀄리 #스칼라 서브쿼리 #인라인 뷰

1. 서브쿼리(SubQuery)란?

하나의 SQL문 안에 포함되어 있는 또 다른 SQL문을 말함.

[그림 Ⅱ-2-12] 메인쿼리와 서브쿼리

2. 서브쿼리 특징

1) 조인의 경우 조인에 참여하는 모든 테이블이 대등한 관계에 있기 때문에 조인에 참여하는 모든 테이블의 컬럼을 어느 위치에서라도 자유롭게 사용

2) 서브쿼리의 경우 메인쿼리의 컬럼을 모두 사용 할 수 있지만 메인쿼리는 서브쿼리의 컬럼을 사용 할 수 없음.

ex.

```
SELECT
    A.USER_ID,
    A.USER_EAMIL,
    B.USER_NAME -- 서브쿼리의 컬럼(사용 x)
FROM USER A
WHERE USER_ID = (
                SELECT
                    B.USER_ID
                FROM USER_DETAIL B
                WHERE A.USER_ID = B.USER_ID -- A.USER_ID는 메인쿼리의 컬럼
                )
```

▲ 위와 같이 서브쿼리에서 사용한 B 테이블을 메인 쿼리에서 사용할 수 없음. (에러 발생)

3) 질의 결과에 서브쿼리 컬럼을 표시해야 된다면 조인 방식으로 변환하거나 함수, 스칼라 서브쿼리(Scalar Subquery) 등을 사용해야 한다.

ex.

```
3-1) 조인 방식
SELECT
    A.USER_ID,

    A.USER_EAMIL,
    B.USER_NAME
FROM USER A, USER_DETAIL B
WHERE A.USER_ID = B.USER_ID

3-2)스칼라 서브쿼리
SELECT
A.USER_ID
A.USER_EMAIL,
(SELECT B.USER_NAME FROM USER_DATIL B WHERE A.USER_ID = B.USER_ID)AS USER_NAME
FORM USER A
```

▲ 위와 같이 서브쿼리에서 사용한 B 테이블을 메인 쿼리에서 사용할 수 없음. (에러 발생)

참고) 서브쿼리 명칭

- SLEECT 문에 있는 서브쿼리 : 스칼라 서브쿼리

- FROM 절에 있는 서브쿼리 : 인라인 뷰

- WHERE 절에 있는 서브쿼리 : 서브쿼리

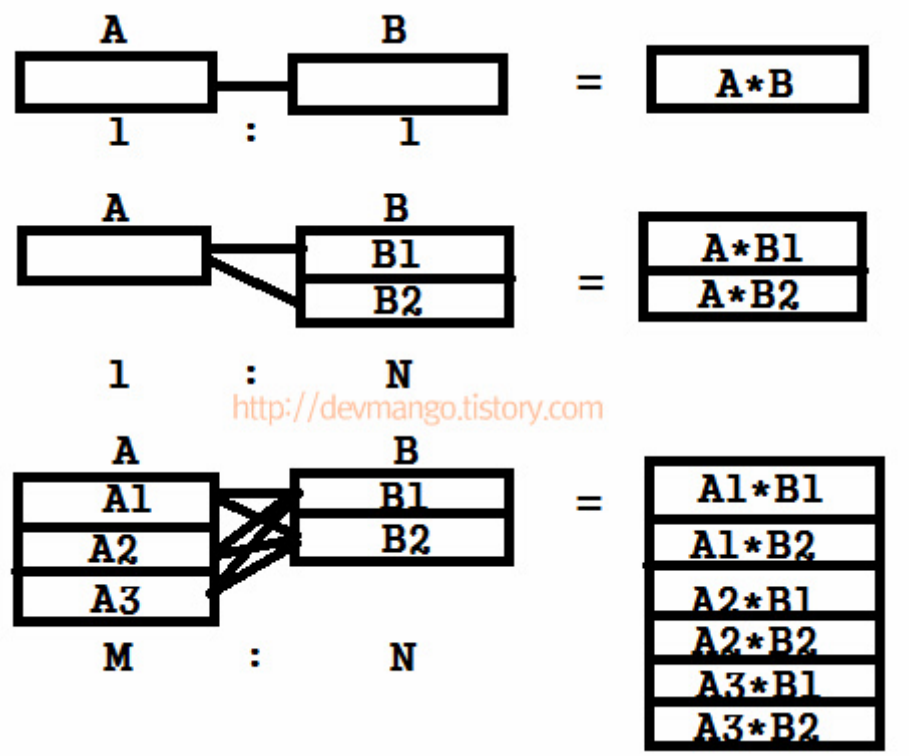
4) 조인은 집합간의 곱(Product)의 관계

4-1) 1:1 관계의 테이블을 조인하면 1(= 1*1) 레벨의 집합이 생성

4-2) 1:M 관계의 테이블을 조인하면 M(= 1*M) 레벨의 집합이 생성

4-3) M:N 관계의 테이블을 조인하면 MN(= M*N) 레벨의 집합이 생성

ex.



5) 서브쿼리는 서브쿼리 레벨과는 상관없이 항상 메인쿼리 레벨로 결과를 생성

ex.

부서 테이블(메인쿼리로 사용)

부서번호	부서명
B001	경영팀
B002	개발팀

사원 테이블(서브쿼리로 사용)

사원번호	사원이름	부서번호
S001	홍길동	B001
S002	홍길순	B002
S003	잡스	B002

5-1) 서브쿼리 방식을 사용 한 경우

```
SELECT
     A.부서번호
    ,A.부서명
FORM 부서 A
WHERE A.부서번호 IN (SELECT B.부서번호 FROM 사원 B WHERE A.부서번호 = B.부서번호)
```

▲ 위와 같이 서브쿼리를 사용하면 결과 집합은 부서 레벨이 됨,

5-1-1) 결과

부서번호	부서명
B001	경영팀
B002	개발팀

5-2) 잘못 된 판단으로 조인 방식을 사용 한 경우

```
SELECT
      A.부서번호
     ,A.부서명
FORM 부서 A, 사원 B
WHERE A.부서번호 = B.부서번호
```

▲ 위와 같이 잘못 된 판단으로 조인을 사용하면 결과 집합은 사원 레벨이 됨.

5-2-2) 결과

부서번호	부서명
B001	경영팀
B002	개발팀
B002	개발팀

5-2-3) 5-1-1과 같이 부서레벨로 만드려면 DISTINCT를 사용해서 중복 데이터가 있는 경우 하나만 들고 오게 수정해야 됨.

3. 서브쿼리 사용 시 주의사항

1) 서브쿼리는 괄호로 감싼 후 사용

2) 서브쿼리는 단일 행(Single Row) 또는 복수 행(Multiple Row) 비교 연산자와 함께 사용 가능

2-1) 단일 행 비교 연산자는 서브쿼리의 결과가 반드시 1건 이하이어야 하고 복수 행 비교 연산자는 서브쿼리의 결과 건수와 상관 없음.

3) 서브쿼리에서는 ORDER BY를 사용하지 못한다. ORDER BY절은 SELECT절에서 오직 한 개만 올 수 있기 대문에 ORDER BY 절은 메인쿼리의 마지막 문자에 위치

4. SQL문에서 사용 가능한 서브쿼리 위치

1) SELECT 절

2) FROM 절

3) WHERE 절

4) HAVING 절

5) ORDER BY 절

6) INSERT문의 VALUES

7) UPDATE문의 SET

5.서브쿼리 종류

[표 Ⅱ-2-4] 동작하는 방식에 따른 서브쿼리 분류

서브쿼리 종류	설명
Un-Correlated(비연관) 서브쿼리	서브쿼리가 메인쿼리 칼럼을 가지고 있지 않는 형태의 서브쿼리이다. 메인쿼리에 값(서브쿼리가 실행된 결과)을 제공하기 위한 목적으로 주로 사용한다.
Correlated(연관) 서브쿼리	서브쿼리가 메인쿼리 칼럼을 가지고 있는 형태의 서브쿼리이다. 일반적으로 메인쿼리가 먼저 수행되어 읽혀진 데이터를 서브쿼리에서 조건이 맞는지 확인하고자 할 때 주로 사용된다.

1) Un-Correlated 서브쿼리 : 서브쿼리에서 메인쿼리 컬럼을 사용하지 않음.

(Un-Correlated Subquery로 검색하면 잘 안 나오고 Non-Correlated Subquery로 검색해야 잘 나옴.)

ex.

```
SELECT
     A.부서번호
    ,A.부서명
FORM 부서 A
WHERE A.부서번호 IN (SELECT B.부서번호 FROM 사원 B) -- 서브쿼리에서 메인쿼리에서 사용된 A테이블의 컬럼을 사용히
```

2) Correlated 서브쿼리 : 서브쿼리에서 메인쿼리 컬럼을 사용 함.

ex.

```
SELECT
     A.부서번호
    ,A.부서명
FORM 부서 A
WHERE A.부서번호 IN (SELECT B.부서번호 FROM 사원 B WHERE B.USER_ID = A.USER_ID) -- 서브쿼리에서 메인쿼리에서
```

6. 서브쿼리 분류

서브쿼리는 메인쿼리 안에 포함된 종속적인 관계이기 때문에 논리적인 실행순서는 항상 메인쿼리에서 읽혀진 데이터에 대해 서브쿼리에서 해당 조건이 만족하는지 확인하는 방식으로 수행되어야 함.

하지만 실제 서브쿼리 실행순서는 상황에 따라 달라짐.

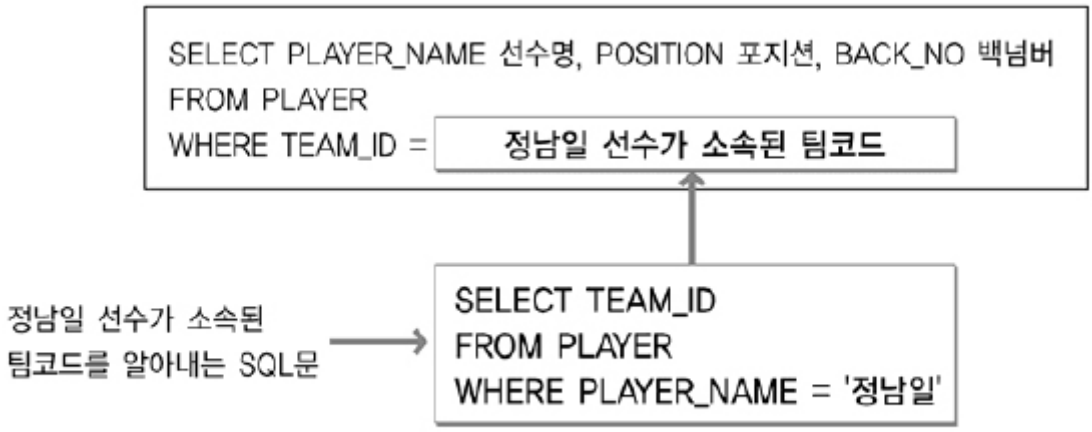
[표 Ⅱ-2-5] 반환되는 데이터의 형태에 따른 서브쿼리 분류

서브쿼리 종류	설명
Single Row 서브쿼리 (단일 행 서브쿼리)	서브쿼리의 실행 결과가 항상 1건 이하인 서브쿼리를 의미한다. 단일 행 서브쿼리는 단일 행 비교 연산자와 함께 사용된다. 단일 행 비교 연산자에는 =, 〈, 〈=, 〉, 〉=, ◇이 있다.
Multi Row 서브쿼리 (다중 행 서브쿼리)	서브쿼리의 실행 결과가 여러 건인 서브쿼리를 의미한다. 다중 행 서브쿼리는 다중 행 비교 연산자와 함께 사용된다. 다중 행 비교 연산자에는 IN, ALL, ANY, SOME, EXISTS가 있다.
Multi Column 서브쿼리 (다중 칼럼 서브쿼리)	서브쿼리의 실행 결과로 여러 칼럼을 반환한다. 메인쿼리의 조건절에 여러 칼럼을 동시에 비교할 수 있다. 서브쿼리와 메인쿼리에서 비교하고자 하는 칼럼 개수와 칼럼의 위치가 동일해야 한다.

▲ 반환되는 데이터 형태에 따라 세 가지로 분류

7. 단일 행 서브 쿼리

1) 서브쿼리가 단일 행 비교 연산자(=, <, <=, >, >=, <>)를 사용할 때는 서브쿼리 결과 건수가 반드시 1건 이하이어야 함. 만약, 2건 이상 반환하면 실행시간(Run Time) 오류 발생함.



[그림 Ⅱ-2-13] 단일 행 서브쿼리의 예제1

ex1.

```
SELECT
    PLAYER_NAME 선수명
    , POSITION 포지션
    , BACK_NO 백넘버
FROM PLAYER
WHERE TEAM_ID = (
                    SELECT
                        TEAM_ID
                    FROM PLAYER
                    WHERE PLAYER_NAME = '정남일'
                )
```

```
ORDER BY PLAYER_NAME;
```

▲ 정남일 선수의 소속팀을 알아내는 서브쿼리가 먼저 수행된 뒤 정남일 선수의 소속팀 코드가 반환 됨.

그리고 메인쿼리는 서브쿼리에서 반환된 결과를 이용해서 조건을 만족하는 선수들의 정보를 출력 함.

만약, 정남일 선수가 동명이인이여서 2건 이상의 결과가 반환되었다면 오류가 발생 될 것임.

ex2.

WHERE TEAM_ID = '256' (O) > 비교연산자는 반드시 단일행

WHERE TEAM_ID = '256','226' (X) > ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다. > WHERE TEAM_ID IN ('256','226') 은 가능

2) 테이블 전체에 하나의 그룹함수를 적용할 때는 그 결과값이 1건만 생성되기 때문에 단일 행 서브쿼리로서 사용 가능함.

[그림 Ⅱ-2-14] 단일 행 서브쿼리의 예제2

ex1.

```
SELECT
      PLAYER_NAME 선수명
      , POSITION 포지션
      , BACK_NO 백넘버
FROM PLAYER
WHERE HEIGHT <= (
                        SELECT
                              AVG(HEIGHT)
                        FROM PLAYER
                       )
ORDER BY PLAYER_NAME;
```

▲ 선수들의 평균키를 알아내는 SQL문(서브쿼리 부분)과 이 결과를 이용해서 키가 평균 이하의 선수들의 정보를 출력하는 SQL문(메인쿼리 부분)으로 구성됨.

만약, 그룹함수를 사용한다 하더라도 아래와 같이 다중 컬럼이 반환된다면 에러가 발생 함.

```
SELECT
      PLAYER_NAME 선수명
      , POSITION 포지션
      , BACK_NO 백넘버
FROM PLAYER
WHERE HEIGHT <= (
                        SELECT
                              AVG(HEIGHT), TEAM_ID
                        FROM PLAYER
                       )
ORDER BY PLAYER_NAME; --에러 발생
SELECT
      PLAYER_NAME 선수명
      , POSITION 포지션
      , BACK_NO 백넘버
FROM PLAYER
WHERE (HEIGHT,  TEAM_ID) IN  (
                        SELECT
                              AVG(HEIGHT), TEAM_ID
                        FROM PLAYER
                       )
ORDER BY PLAYER_NAME; --해당 기능은 SQL Server에서는 지원되지 않음.
```

▲ 선수들의 평균키를 알아내는 SQL문(서브쿼리 부분)과 이 결과를 이용해서 키가 평균 이하의 선수들의 정보를 출력하는 SQL문(메인쿼리 부분)으로 구성됨.

8. 다중 행 서브쿼리

서브쿼리의 결과가 2건 이상 반환 될 경우 반드시 다중 행 비교 연산자(IN, ALL, ANY, SOME) 와 함께 사용해야 함.

그렇지 않으면 오류를 반환 함.

만약, 서브쿼리에서 반환 값이 없으면 메인쿼리에서도 반환 값이 없음.

[표 Ⅱ-2-6] 다중 행 비교 연산자

다중 행 연산자	설명
IN (서브쿼리)	서브쿼리의 결과에 존재하는 임의의 값과 동일한 조건을 의미한다. (Multiple OR 조건)
비교연산자 ALL (서브쿼리)	서브쿼리의 결과에 존재하는 모든 값을 만족하는 조건을 의미한다. 비교 연산자로 ">"를 사용했다면 메인쿼리는 서브쿼리의 모든 결과 값을 만족해야 하므로, 서브쿼리 결과의 최대값보다 큰 모든 건이 조건을 만족한다.
비교연산자 ANY (서브쿼리)	서브쿼리의 결과에 존재하는 어느 하나의 값이라도 만족하는 조건을 의미한다. 비교 연산자로 ">"를 사용했다면 메인쿼리는 서브쿼리의 값들 중 어떤 값이라도 만족하면 되므로, 서브쿼리의 결과의 최소값보다 큰 모든 건이 조건을 만족한다. (SOME은 ANY와 동일함)
EXISTS (서브쿼리)	서브쿼리의 결과를 만족하는 값이 존재하는지 여부를 확인하는조건을 의미한다. 조건을 [출처] 서브쿼리의 종류|작성자 찐 만족하는 건이 여러 건이더라도 1건만 찾으면 더 이상 검색하지 않는다.

▲ ANY는 OR의 개념, ALL은 AND 개념과 유사

PLAYER 테이블

PLAYER_NAME	PLAYER_ID	POSITION	BACK_NO	HEIGHT
김수현	001	(NULL)	(NULL)	165
원빈	002	(NULL)	(NULL)	190
박보검	003	(NULL)	(NULL)	178
송중기	004	(NULL)	(NULL)	195
임맹고	005	(NULL)	(NULL)	201
홍길동	006	(NULL)	(NULL)	140

ex1. IN : 서브쿼리가 리턴 하는 행 중 하나라도 만족하는 경우 결과를 리턴

```
SELECT
      PLAYER_NAME 선수명
      , POSITION 포지션
      , BACK_NO 백넘버
      , HEIGHT 키
FROM PLAYER
WHERE HEIGHT IN (
                        SELECT
                            HEIGHT
                        FROM PLAYER
                        WHERE HEIGHT = '165' OR HEIGHT = '190' OR HEIGHT = '178'
                       )
ORDER BY PLAYER_NAME;

-- 서브쿼리 결과 값이 178, 190, 165 이라는 전제
-- HEIGHT = '178' OR HEIGHT = '190' OR HEIGHT =  '165' 와 같음
-- 하지만, IN()의 경우 FULLSCAN을 하기 때문에 속도가 느림.
-- 예를 들면 IN의 경우 178, 190, 165 중 178이 맞아도 다음줄로 안 넘어가고 190, 165를 다 검사함.
```

-- = 와 OR를 사용하면 178 하나만 맞으면 다음줄로 넘어감.
(이 부분은 발표때 추가 설명 하겠음!!)

ex1-1. 결과

선수명	포지션	백넘버	키
김수현	(NULL)	(NULL)	165
박보검	(NULL)	(NULL)	178
원빈	(NULL)	(NULL)	190

ex2. ALL 비교 연산자(AND 개념)

```
SELECT
      PLAYER_NAME 선수명
      , POSITION 포지션
      , BACK_NO 백넘버
      , HEIGHT 키
FROM PLAYER
WHERE HEIGHT > ALL(
                        SELECT
                              HEIGHT
                        FROM PLAYER
                        WHERE HEIGHT = '165' OR HEIGHT = '190' OR HEIGHT = '178'
                       )
ORDER BY PLAYER_NAME;

-- 서브쿼리 결과 값이 178, 190, 165 으로 다중반환 된 경우 > ALL은 이 중 최대값보다 크다는 것을 의미 / < ALL 은 최소
-- 서브쿼리에서 리턴되는 모든 값을 만족하면 조건이 성립되기 때문에, 서브쿼리 결과 값 중에서 최대값으로 비교하면 모
-- > ALL : 178보다 크고, 165보다 크고, 190보다 크다라는 AND 조건이 성립
-- < ALL : 190 보다 작고, 178보다 작고, 165보다 작다라는 AND 조건이 성립
```

ex2-1. > ALL 결과

선수명	포지션	백넘버	키
송중기	(NULL)	(NULL)	195
임맹고	(NULL)	(NULL)	201

ex3. ANY 비교 연산자(OR 개념)

```
SELECT
      PLAYER_NAME 선수명
      , POSITION 포지션
      , BACK_NO 백넘버
FROM PLAYER
WHERE HEIGHT > ANY(
                        SELECT
```

```
                    HEIGHT
                FROM PLAYER
            )
ORDER BY PLAYER_NAME;

-- 서브쿼리 결과 값이 178, 190, 165 으로 다중반환 된 경우 > ANY는 이 중 어느 하나라도 만족하면 되므로 최소값 165보
-- 즉, 현재 비교연산자로 >를 사용했으므로 165보다 큰 HEIGHT 데이터를 찾음 / <를 사용했으면 190보다 작은 데이터를
```

ex3-1.

선수명	포지션	백넘버	키
박보검	(NULL)	(NULL)	178
송중기	(NULL)	(NULL)	195
원빈	(NULL)	(NULL)	190
임맹고	(NULL)	(NULL)	201

ex4. = ANY 연산자와 IN 연산자는 동일 / ANY 와 SOME 은 동일

ex5. <>ANY 연산자는 NOT IN 과 다름 : <>ANY의 경우 NOT = A OR NOT = B OR NOT = C 를 의미하지만 IN은 NOT = A AND NOT = B AND NOT = C를 의미

ex6. <>ALL 연산자는 NOT IN 과 동일

PLAYER 테이블

PLAYER_NAME	TEAM_ID	PLAYER_ID	POSITION	BACK_NO	HEIGHT
원빈	T004	002	(NULL)	(NULL)	190
송중기	T003	004	(NULL)	(NULL)	195
임맹고	T002	005	(NULL)	(NULL)	201
홍길동	T002	006	(NULL)	(NULL)	140
김수현	T001	001	(NULL)	(NULL)	165
박보검	T001	003	(NULL)	(NULL)	178
김땡땡	(NULL)	007	(NULL)	(NULL)	180

TEAM_INFO 테이블

TEAM_ID	TEAM_NAME
T001	잘생긴팀
T002	젊은팀
(NULL)	돈많은팀

ex7. EXISTS검색된 결과가 하나라도 존재하면 메인쿼리 조건절이 참

(EXISTS 처럼 서브쿼리에 메인쿼리 컬럼이 사용되면 연관 서브쿼리(Correlated Subquery)라고 함.)

```
-- EXISTS() 사용
SELECT
    PLAYER_NAME 선수명
    , POSITION 포지션
    , BACK_NO 백넘버
```

```
    , HEIGHT 키
    , TEAM_ID 팀아이디
FROM PLAYER P
WHERE EXISTS (
                SELECT
                    *
                FROM TEAM_INFO T
                WHERE  T.TEAM_ID = P.TEAM_ID
                 )
ORDER BY PLAYER_NAME ;

--IN() 사용
SELECT
    PLAYER_NAME
    ,POSITION
    ,BACK_NO
    ,HEIGHT
    ,TEAM_ID
FROM PLAYER P
WHERE TEAM_ID IN (
                    SELECT
                        TEAM_ID
                    FROM TEAM_INFO T
                    )
ORDER BY PLAYER_NAME;
```

ex7-1. EXISTS() 사용 결과와 IN() 사용 결과가 같음.

PLAYER_NAME	POSITION	BACK_NO	HEIGHT	TEAM_ID
김수현	(NULL)	(NULL)	165	T001
박보검	(NULL)	(NULL)	178	T001
임맹고	(NULL)	(NULL)	201	T002
홍길동	(NULL)	(NULL)	140	T002

ex8. NOT EXISTS() 는 서브쿼리에서 검색된 결과가 하나도 존재하지 않으면 메인쿼리 조건절이 참

```
-- NOT EXISTS() 사용
SELECT
    PLAYER_NAME 선수명
    , POSITION 포지션
    , BACK_NO 백넘버
    , HEIGHT 키
    , TEAM_ID 팀아이디
FROM PLAYER P
WHERE NOT EXISTS (
                SELECT
                    *
```

```
                FROM TEAM_INFO T
                WHERE  T.TEAM_ID = P.TEAM_ID
                )
ORDER BY PLAYER_NAME ;

--NOT IN() 사용
SELECT
    PLAYER_NAME
    ,POSITION
    ,BACK_NO
    ,HEIGHT
    ,TEAM_ID
FROM PLAYER P
WHERE TEAM_ID NOT IN (
                        SELECT
                            TEAM_ID
                        FROM TEAM_INFO T
                        )
ORDER BY PLAYER_NAME;
```

ex8-1. NOT EXISTS 결과(OR)

PLAYER_NAME	POSITION	BACK_NO	HEIGHT	TEAM_ID
김땡땡	(NULL)	(NULL)	180	(NULL)
송중기	(NULL)	(NULL)	195	T003
원빈	(NULL)	(NULL)	190	T004

▲ TEAM_INFO 테이블에 저장된 TEAM_ID의 경우 T001, T002, NULL 이니 일반적으로 T001, T002, NULL이 아닌 T003, T004을 팀으로 가지는 송중기, 원빈만 데이터로 나올 것이라고 생각할 수 있음.

하지만 결과를 보면 TEAM_ID가 NULL인 김땡땡도 보여짐.

EXISTS의 경우는 NULL 이 안 나오는데 NOT EXISTS의 경우는 NULL도 나와서 찾아본 바로는

1. NULL = NULL 이면 false 이니(NULL끼리 비교는 안 되므로) 당연히 안 보여져야 되는데 NOT 으로 인해 true 가 되니 메인 쿼리가 null도 보여지게 되는 거라고 함.

(2개의 테이블에 null만 있는 데이터 1개씩 넣어서 테스트 해보면 조금 더 이해가 됨.)

내가 이해한대로라면 NOT EXISTS (001, 002, FALSE(NULL)) 가 되어 결과는 003 OR 004 OR TURE(NULL) 가 되어 위의 결과가 보여지는 듯 함.

ex8-2. NOT IN 결과(AND)

PLAYER_NAME	POSITION	BACK_NO	HEIGHT	TEAM_ID

▲ NOT IN의 경우도 T001, T002, NULL 이 포함되지 않은 데이터 T003, T004를 팀으로 가지는 송중기, 원빈이 나올 것이라고 생각하게 됨.

하지만 0건의 데이터가 결과로 나옴.

혹시나 해서 TEAL_INFO 테이블에 TEAM_ID가 NULL 인 데이터에 T006을 넣어줬더니 원하는 결과가 나왔음.

풀어서 써보면 TEAM_ID <> T001 AND TEAM_ID <> T002 AND TEAM_ID <> NULL 인데 NULL이 아닌 경우는 IS NOT NULL을 써야되므로 TEAM_ID <> NULL는 당연히 문법적으로 틀렸기 때문에 1건도 검색하지 못 하게 됨.
반대로 생각해서 그럼 IN은 왜 NULL 데이터가 있어도 검색이 되지? 라고 생각할 수 있는데 IN은 OR 조건이기 때문에 하나라도 TRUE가 되면 서브쿼리 자체가 TRUE로 반환되므로 나오는 것임.

ex8-3. 결론
IN()은 조건에 만족하는 데이터를 찾는 것이고 EXISTS는 TRUE인지 FALSE인지를 체크하는 것임.
IN의 경우 JOIN 연산을 하지 않고, EXISTS는 JOIN 연산을 함.
그러므로 NULL 값을 가질 때 NOT IN의 경우 값을 비교하는거다보니 값 <> NULL이 되어 잘못된 문법으로 결과에서 아예 제외되버리고,
EXISTS의 경우 값을 만족하는지 안 하는지에 대해서만 체크를 하고 데이터를 비교하는게 아니기 때문에 FALSE를 리턴하고 FALSE의 NOT이니까 결국 TURE가 되어 NULL도 결과에 포함됨.
만약 NOT IN에서 NULL값도 비교하게 하고 싶다면 NVL을 이용하여 NULL 처리를 해줘야 됨.

9. 그 밖에 위치에서 사용하는 서브쿼리
1) SELECT 절에 서브쿼리 사용하기(스칼라 쿼리) : 한 행, 한 컬럼만을 반환해야 함 / 2건 이상 반환되면 오류 발생

```
SELECT
    PLAYER_NAME
    ,HEIGHT
    ,(SELECT AVG(HEIGHT) FROM PLAYER X WHERE X.TEAM_ID = P.TEAM_ID)AS AVG_HEIGHT
FROM PLAYER P
```

2) FROM 절에서 서브쿼리 사용하기(인라인 뷰) :
- 인라인 뷰는 SQL문이 실행될 때만 임시적으로 생성되는 동적인 뷰이기 때문에 데이터베이스에 해당 정보가 저장되지 않음.
- 일반적인 뷰를 정적 뷰(Static View)라고 하고, 인라인 뷰를 동적 뷰(Dynamic View)라고 함.
- 인라인 뷰는 테이블 명이 올 수 있는 곳에서 사용 가능
- 서브쿼리의 컬럼은 메인쿼리에서 사용 할 수 없으나 인라인 뷰는 동적으로 생성된 테이블임.
- 인라인 뷰를 사용하는 것은 조인 방식을 사용하는 것과 같기 때문에 인라인 뷰의 컬럼을 SQL문에서 자유롭게 참조할 수 있음.

```
SELECT
    T.TEAM_NAME
    ,P.PLAYER_NAME
    ,P.BACK_NO
FROM
    (
        SELECT TEAM_ID
                ,PLAYER_NAME
                ,BACK_NO
```

```
        FROM PLAYER
        WHERE POSITION = 'MF'
    ) P
    , TEAM T
WHERE P.TEAM_ID = T.TEAM_ID
ORDER BY TEAM_NAME;

-- TOP-쿼리 : 인라인 뷰에서 먼저 정렬을 수행하고 정렬된 결과 중에서 일부 데이터를 추출하는 작업
SELECT
    PLAYER_NAME
    BACK_NO
FROM
    (
        SELECT TEAM_ID
                ,PLAYER_NAME
                ,BACK_NO
        FROM PLAYER
        WHERE HEIGHT IS NOT NULL
        ORDER BY HEIGHT DESC
    )
    WHERE ROWNUM <= 5;
```

3) HAVING 절에서 서브쿼리 사용하기 : GROUP함수를 사용할 때 그룹핑 된 결과에 대해 부가적인 조건을 줄 때 사용

```
SELECT
    P.TEAM_ID
    ,T.TEAM_NAME
    ,AVG(P.HEIGHT) AS AVG_HEIGHT
FROM
    PLAYER P
    , TEAM T
WHERE P.TEAM_ID = T.TEAM_ID
GROUP BY P.TEAM_ID
            ,T.TEAM_NAME
HAVING AVG(P.HEIGHT) < (SELECT AVG(HEIGHT) FROM PLAYER WHERE TEAM_ID = 'K02')
```

4) UPDATE문의 SET 절에서 사용하기 : 서브쿼리 결과가 NULL을 반환할 경우 해당 컬럼의 결과가 NULL이 될 수 있기 때문에 주의 필요

```
UPDATE
    TEAM A
SET
    A.STADIUM_NAME = (
```

```
            SELECT
                X.STADIUM_NAME
            FROM STADIUM X
            WHERE X.STADIUM_ID = A.STADIUM_ID
        )
```

5) INSERT 문의 VALUES 절에서 사용하기

```
INSERT INTO
    PLAYER
    (
      PLAYER_ID
    , PLAYER_NAME
    , TEAM_ID
    )
    VALUES
    (
    (SELECT TO_CHAR(MAX(TO_NUMBER(PALYER_ID))+1) FROM PLAYER)
    ,'홍길동'
    ,'K06'
    );

-- 테이블 복사 할때도 사용(데이터 복사)
CREATE TABLE TEMP_PLAYER
AS SELECT * FROM PLAYER;

-- mysql의 경우 테이블 구조까지 복사되지 않음 / 테이블 구조도 복사하는 방법
CREATE TEMP_PLAYER LIKE PLAYER ;--테이블 및 테이블 구조 복사 후
INSERT INTO TEMP_PLAYER SELECT * FROM PLAYER ;--  데이터 복사

-- 테이블만 복사
CREATE TABLE TEMP_PLAYER
AS SELECT * FROM PLAYER WHERE 1 = 2 ;--조건이 성립하지 않게 해서 데이터는 복사하지 않음.
```

10. 뷰(VIEW)

- 테이블은 실제 데이터를 가지고 있는 반면, 뷰는 실제 데이터를 가지고 있지 않음.
- 뷰는 단지 뷰 정의만을 가지고 있음.
- SQL문에서 뷰를 사용하면 DBMS 내부적으로 뷰 정의를 참고해 재작성하여 SQL문을 수행함.
- 가상 테이블이라고 함.

```
CREATE VIEW
    V_PLAYER_TEAM
AS
```

```
    SELECT
        P.PLAYER_NAME
        , P.POSITION
        , P.BACK_NO
        , P.TEAM_ID
        , T.TEAM_NAME
    FROM PLAYER P, TEAM T
    WHERE P.TEAM_ID = T.TEAM_ID;

-- 이미 존재하는 뷰를 참조해서 뷰 생성 가능 / 하지만 뷰를 포함하는 뷰를 잘 못 생성하는 경우 성능상의 문제를 유발 할
CREATE VIEW
    V_PLAYER_TEAM_FILTER
AS
    SELECT * FROM V_PLAYER_TEAM
    WHERE POSITION IN('GK','MF');

-- 뷰를 통해 데이터 조회
SELECT PLAYER_NAME, TEAM_NAME FROM V_PLAER_TEAM WHERE PLAYER_NAME LIKE '%맹고%';

-- 뷰를 통해 조회하면 아래와 같이 DBMS가 내부적으로 SQL문을 재작성함.
SELECT
    PLAYER_NAME
    , TEAM_NAME
FROM
    (
    SELECT
        P.PLAYER_NAME
        , P.POSITION
        , P.BACK_NO
        , P.TEAM_ID
        , T.TEAM_NAME
    FROM PLAYER P, TEAM T
    WHERE P.TEAM_ID = T.TEAM_ID
    )
WHERE P.PLAYER_NAME LIKE '%맹고%';

--뷰 제거
DROP VIEW V_PLAYER_TEAM;
```

참고)

- 뷰는 조회만 가능한 줄 알았는데, 찾아보니 DML도 가능함.

다만 테이블 하나만 사용(단순뷰)해서 생성한 뷰일 경우만 가능하고 테이블 두개 이상(복합뷰) 사용시에는 DML 안 됨.

- DML을 사용하지 못하게 하려면 "WITH READ ONLY" 옵션을 사용하면 되는데 해당 옵션을 사용해서 생성한 뷰에서 DML을 사용하면 에러 발생

- 중요한 점은 뷰의 데이터를 수정하게 되면 실제 테이블의 데이터가 수정 됨. > 테이블이 수정되면 뷰 또한 수정된 값을 보여줌.

* DB 스터디 참고 교재 : SQL 개발자 가이드

* 포스팅 참고 URL :

1) http://www.dbguide.net/db.db?cmd=view&boardUid=148203&boardConfigUid=9&categoryUid=216&boardIdx=135&boardStep=1

2) http://ttend.tistory.com/623

3)https://technet.microsoft.com/ko-kr/library/ms187074(v=sql.105).aspx

4)http://www.kkomzi.net/wordpress/128

5)http://tystory.tistory.com/211

6)http://tipland.tistory.com/6

'DBMS > DBMS스터디' 카테고리의 다른 글

SQL 최적화 기본 원리] 인덱스 기본 (0)	2018.01.01
SQL 최적화 기본 원리] 옵티마이저와 실행계획 (0)	2018.01.01
[SQL활용] 그룹 함수 (0)	2017.12.06
[SQL 활용] 서브쿼리 (3)	2017.12.06

http://

SECRET WRITE

2017.12.06 22:23

비밀댓글입니다

Delete Reply

박동운 2017.12.06 23:17 신고

알기쉽게 정리해주셔서 정말 감사합니다!!

Delete Reply

개발하는맹고언니 달달하션 2017.12.09 21:35 신고

^^

Delete

PREV 1 … 6 7 8 9 10 11 12 13 14 … 26 NEXT

+ Recent posts

table의 tbody 영역에만 스크..

테이블에서 글자수 많으면 .....

SQL 쿼리 편집기에서 항상 Ctr..

오라클 테이블스페이스 생성