이때까지 배운 내용을 한번 정리해볼까요?

두 테이블을 서로 합치는 연산에는 크게 두 가지 종류가 있다고 했습니다.

첫 번째는 두 테이블을 가로 방향으로 합치는 것에 관한 **결합** 연산,

두 번째는 두 테이블을 세로 방향으로 합치는 것에 관한 **집합** 연산

이라고 했는데요.

결합 연산 중에서는 LEFT OUTER JOIN, RIGHT OUTER JOIN, INNER JOIN

집합 연산 중에서는 INTERSECT, MINUS, UNION, UNION ALL

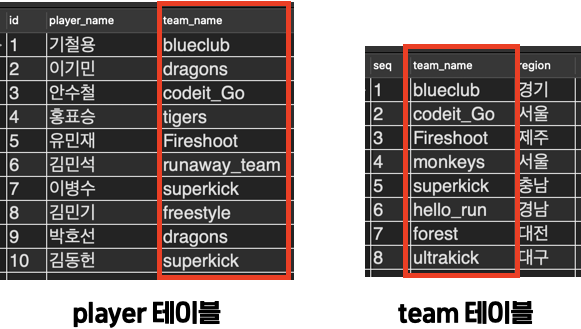
을 배웠습니다. 이때 집합 연산 중 INTERSECT, MINUS 연산자는 MySQL에서 지원하지 않아서, 조인을 통해 간접적으로 원하는 결과를 얻었던 거, 기억나시죠?

이번 노트에서는 우리가 배우지 않았던 조인 종류들을 배워보겠습니다. 이전에 배운 것들에 비해 실무적으로 활용도는 떨어지지만, 알아두고 있으면 좋은 내용이라서 소개하겠습니다. 혹시라도 현장에서 일을 하다가 이번 노트에서 나온 조인을 들었는데 모르는 상태면 안 될 테니까요.

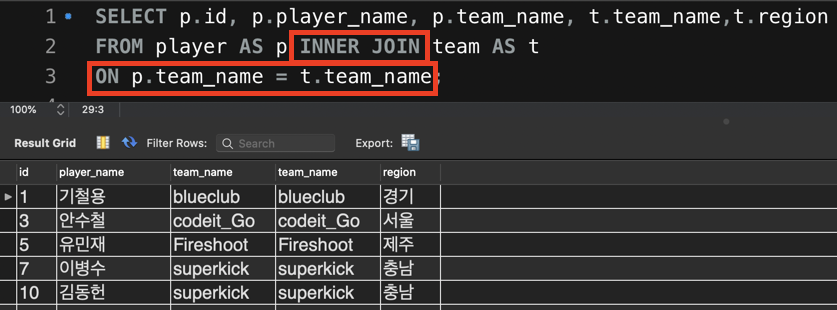
하나씩 순서대로 소개하겠습니다.

**1. NATURAL JOIN**

아래와 같이 축구 선수 정보가 담긴 player 테이블과 축구팀 정보가 담긴 team 테이블이 있다고 해봅시다.

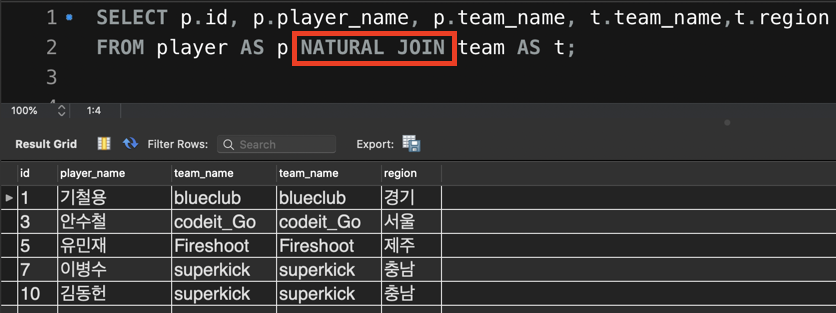


지금 두 테이블에는 team\_name이라는, 같은 이름의 컬럼이 있는데요. 이 두 컬럼을 기준으로 조인을 하면 축구 선수가 속한 축구팀과 축구팀의 기반 지역을 한눈에 볼 수 있을 것 같습니다. 두 테이블을 INNER JOIN 해보겠습니다.



우리가 예상한 대로 조인이 잘 됩니다.

그런데 이런 식의 조인도 가능합니다.



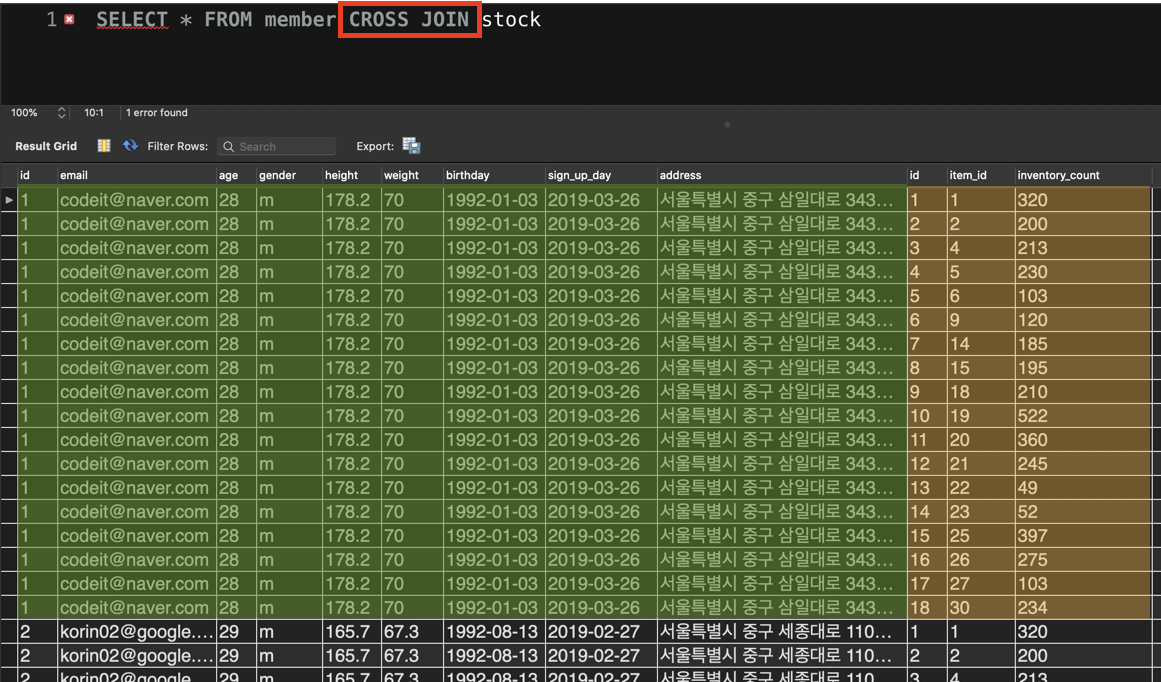
저는 **INNER JOIN이라고 쓴 부분을 NATURAL JOIN으로 바꾸었고, 조인 조건을 나타내는 ON 절을 아예 삭제해버렸는데요.** NATURAL JOIN이 뭘까요? NATURAL JOIN은 두 테이블에서 같은 이름의 컬럼을 찾아서 자동으로 그것들을 조인 조건을 설정하고, INNER JOIN을 해주는 조인입니다. 우리말로는 자연 조인이라고도 하는데요.

이때까지 조인을 할 때마다 조인 조건을 설정했던 것과는 달리 NATURAL JOIN은 조인 조건을 자동으로 설정해주기 때문에 ON 절을 쓸 필요가 없습니다. 단어 뜻 그대로 별도의 조인 조건 설정 없이, **자연**스럽게 진행되는 조인인 거죠.

사실 두 테이블에 같은 이름의 컬럼이 있더라도 NATURAL JOIN을 쓰기보다는 우리가 배운 조인을 쓰고 ON 절에 조인 조건을 명시해주는 것이 좋습니다. NATURAL JOIN을 해버리면 SQL 문을 보더라도, 테이블 구조를 모르는 사람이라면 어떤 컬럼들을 기준으로 조인이 될지 알 수 없으니까요. 하지만 NATURAL JOIN이 사용된 SQL 문을 만나게 되면, 해석할 수 있어야하기 때문에 알려드리는 겁니다.

**2. CROSS JOIN**

CROSS JOIN은 한 테이블의 하나의 row에 다른 테이블의 모든 row들을 매칭하고, 그 다음 row에서도 또, 다른 테이블의 모든 row들을 매칭하는 것을 반복함으로써 두 테이블의 row들의 모든 조합을 보여주는 조인입니다. 잠깐 아래 그림을 보세요.



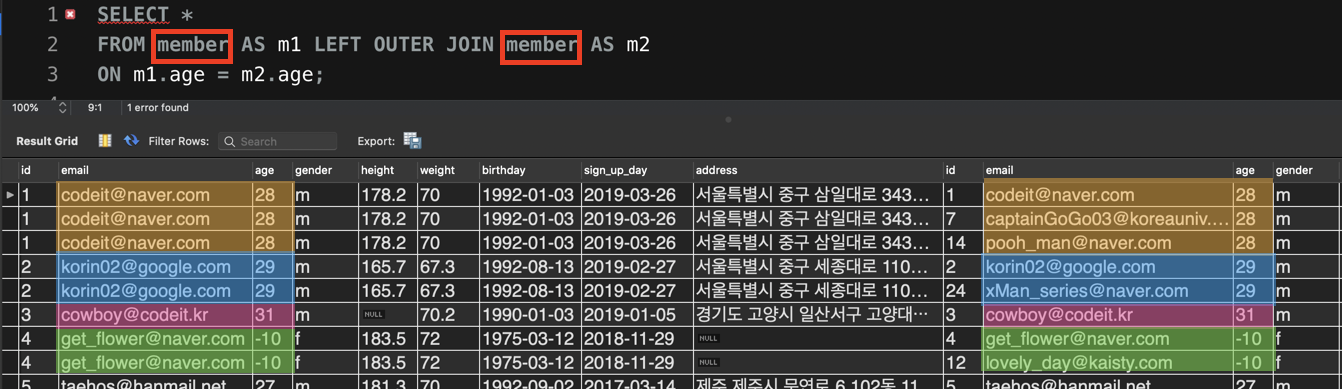
지금 member 테이블과 stock 테이블을 CROSS JOIN 했습니다. 그랬더니

codeit@naver.com 회원을 나타내는 row에 stock 테이블의 모든 row들이 매칭되어서 표시되었죠? 그리고 그 아래에는 또 그 다음 회원에 대해서 같은 작업을 한 결과가 표시되고, 그 다음에는 또 그런 결과가 표시되는 식으로 해서 두 테이블의 row들의 모든 조합이 표시됩니다. 테이블 하나를 잠깐 하나의 집합이라고 생각해봅시다. 그럼 각 row는 하나의 원소가 되겠죠? 방금처럼 두 집합의 모든 원소들의 조합을 나타내는 것을 수학의 집합 이론에서는 카르테시안 곱(Cartesian Product)이라고 하는데요. CROSS JOIN은 두 테이블의 Cartesian Product를 구하는 조인인 겁니다.

이 CROSS JOIN은 어떤 경우에 사용할 수 있을까요? 예를 들어, 여러 종류의 의류들 중에서도 상의들의 정보가 담긴 테이블, 하의들의 정보가 담긴 테이블이 있다고 해봅시다. 이때, 옷을 입을 때의 상-하의 조합들을 한눈에 보고싶은 경우에 CROSS JOIN을 사용하면 되겠죠? 하지만 일반적인 경우에는 잘 쓸 일이 없는 종류의 조인이기는 합니다.

**3. SELF JOIN**

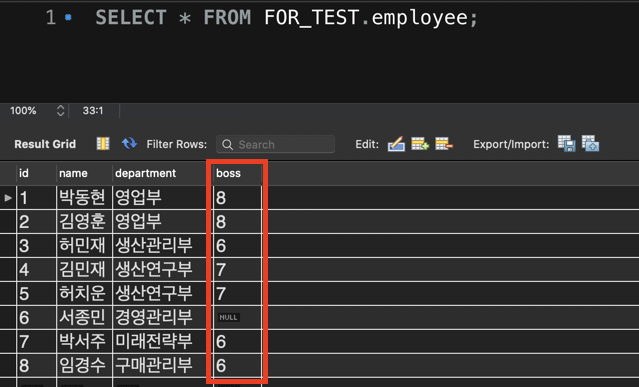
SELF JOIN은 조인 방식에 있어서 새로운 조인은 아닙니다. SELF JOIN은 셀프라는 단어의 뜻 그대로 테이블이 자기 자신과 조인을 하는 경우를 말합니다. SELF JOIN이라고 해서 헷갈릴 필요는 없습니다. 그냥 서로 별개인 두 테이블을 조인하는 것처럼 생각하시면 되는데요. 어떤 경우에 이런 조인이 필요할까요? 아래 그림을 보세요.



지금 저는 member 테이블을 SELF JOIN 하고 있습니다. 지금 같은 테이블이기 때문에 그 이름 구별을 하기 위해서 각각 다른 alias(m1, m2)를 주었습니다. 그리고 두 테이블의 age 컬럼을 조인 기준으로 해서 LEFT OUTER JOIN을 했죠? 그 결과를 보니 어떤가요.

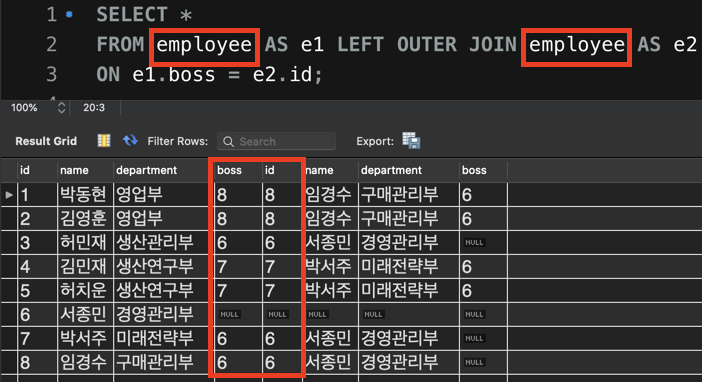
지금 각 회원마다 자신과 동갑인 다른 회원들(본인 포함)이 함께 출력되는 것을 알 수 있습니다. 예를 들어 codeit@naver.com 회원이 지금 28살인데 동갑인 회원으로 captainGOGO03@~, pooh\_man@~ 회원이 있다는 것을 알 수 있네요. 이렇게 SELF JOIN을 통해 하나의 테이블 안에서 다양한 정보들을 추출해볼 수 있습니다.

또다른 예시를 살펴볼까요?



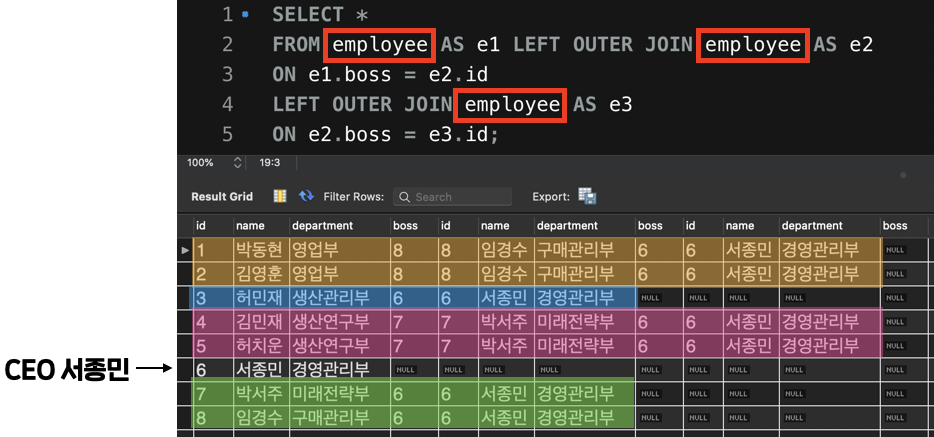
지금 보면 employee라는 테이블에 id(Primary key), name(직원의 이름), department(소속 부서), boss(직속 상사의 id 값) 컬럼이 있습니다. 지금 boss 컬럼의 값들은 결국 같은 테이블의 id 컬럼에 있는 값들 중 하나인데요. 어떤 직원의 직속 상사도 당연히 그 회사의 직원일 테니까 당연한 겁니다.

이 상태에서 잠깐 SELF JOIN을 해볼게요.



employee 테이블의 boss 컬럼과 id 컬럼을 기준으로 LEFT OUTER JOIN인 SELF JOIN을 했더니 각 직원 옆에 직속 상사 정보도 함께 뜨죠?

이 결과에서 같은 방식으로 한번 더 SELF JOIN을 해볼게요.



LEFT OUTER JOIN인 SELF JOIN을 한번 더 하니까, 한 직원의, 직속 상사의, 직속 상사까지도 볼 수가 있습니다. 결과를 보니 경영관리부의 '서종민'이라는 분이 CEO인 것 같네요. 직속 상사 정보가 없고, 다른 직원들 입장에서 가장 멀리 있는 직속 상사인 걸 보니까 말이죠.

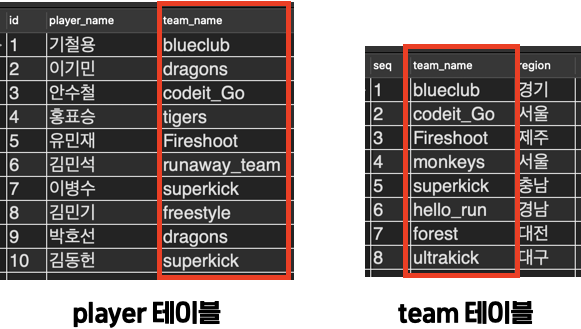
방금 본 것처럼 SELF JOIN은 조인 방식에 있어서 뭔가 새로운 조인은 아닙니다. 다만, 조인 대상이 같은 테이블을 마치 별도의 테이블인 것처럼 간주하고 진행된다는 점에서 특색이 있는 조인인데요.

방금 보신 것처럼 SELF JOIN을 하면 하나의 테이블에 담긴 데이터를 다양한 관점에서 바라볼 수 있게 됩니다.

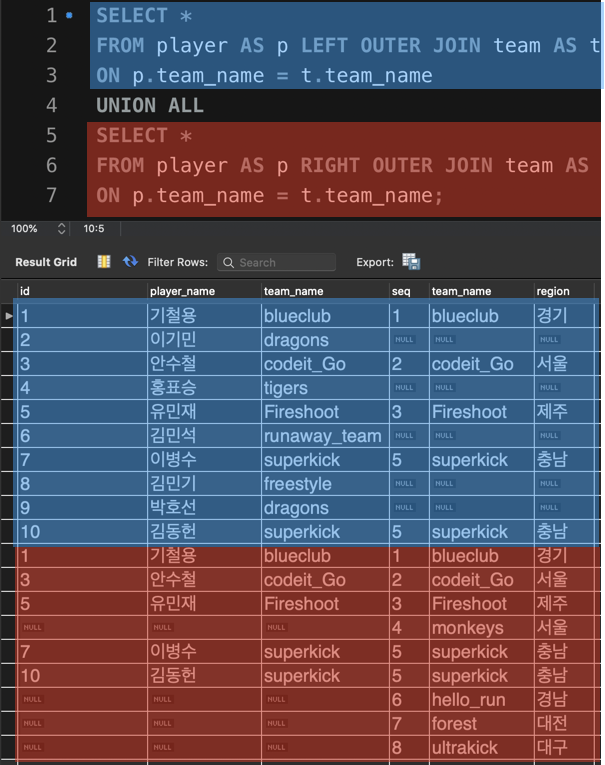
**4. FULL OUTER JOIN**

FULL OUTER JOIN은 뭘까요? 우리는 LEFT OUTER JOIN과, RIGHT OUTER JOIN을 배웠습니다. 둘다 왼쪽이나 오른쪽에 있는 테이블 하나를 기준으로 두고 상대 테이블을 조인하는 거였죠?

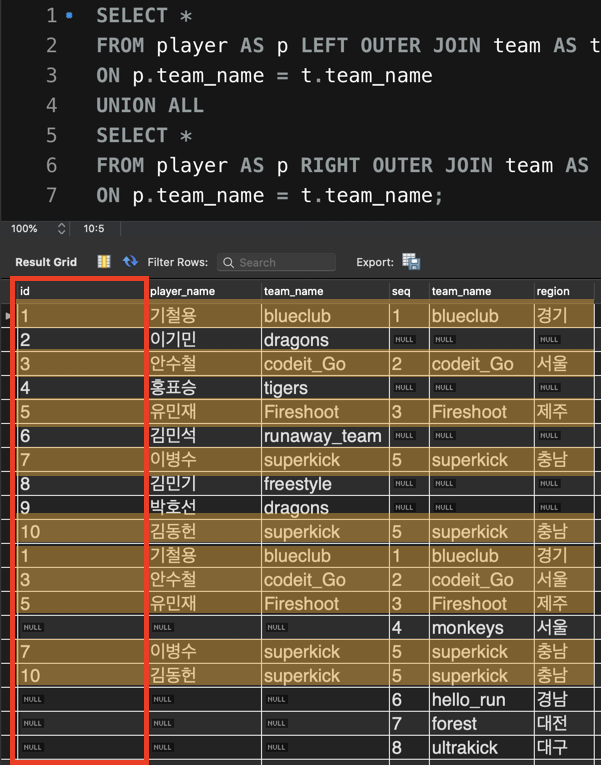
**FULL OUTER JOIN은 두 테이블의 LEFT OUTER JOIN 결과와 RIGHT OUTER JOIN 결과를 합치는 조인**입니다. 대신, 이때 두 결과에 모두 존재하는 row들(두 테이블에 공통으로 존재하던 row들)은 한번만 표현해주죠.



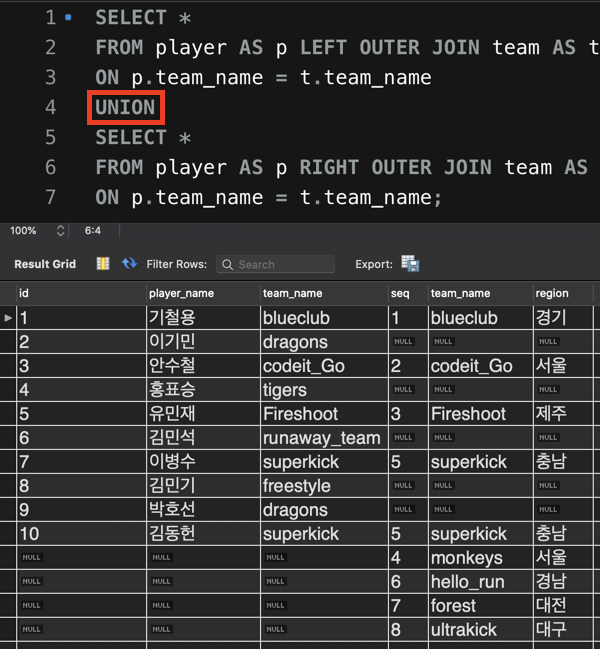
위에서 Natural Join을 배울 때 봤던 이 두 테이블을 FULL OUTER JOIN 해보겠습니다.



위 SQL 문, 이해되시나요? 지금 하늘색 영역이 LEFT OUTER JOIN 부분이고, 그 결과입니다. 그리고 빨간색 영역이 RIGHT OUTER JOIN 부분이고, 그 결과이구요. 이 둘을 UNION ALL한 결과인데요. 두 결과가 그대로 잘 합쳐졌습니다. 그런데 지금 이 결과를 보면



서로 겹치는 row들이 보입니다. id 컬럼을 보면 개수가 2개인 row들이 보이는데요. 이런 중복 row들은 제거하고 하나만 보이도록 해볼게요. 어떻게 하면 될까요? [이전 노트에서 배운대로](https://www.codeit.kr/learn/3256) UNION ALL 대신 UNION을 사용하시면 됩니다.



그럼 이렇게 중복이 제거된 결과가 나타납니다. 이 결과가 바로 FULL OUTER JOIN의 결과인데요. 앞으로 FULL OUTER JOIN이라는 말을 들으면 방금 전 설명을 기억하세요.

참고로, Oracle이라는 DBMS에서는 FULL OUTER JOIN을 바로 할 수 있도록 해주는 연산자가 내장되어 있습니다. 그래서

SELECT \*

FROM player AS p FULL OUTER JOIN team AS t

ON p.team\_name = t.team\_name;

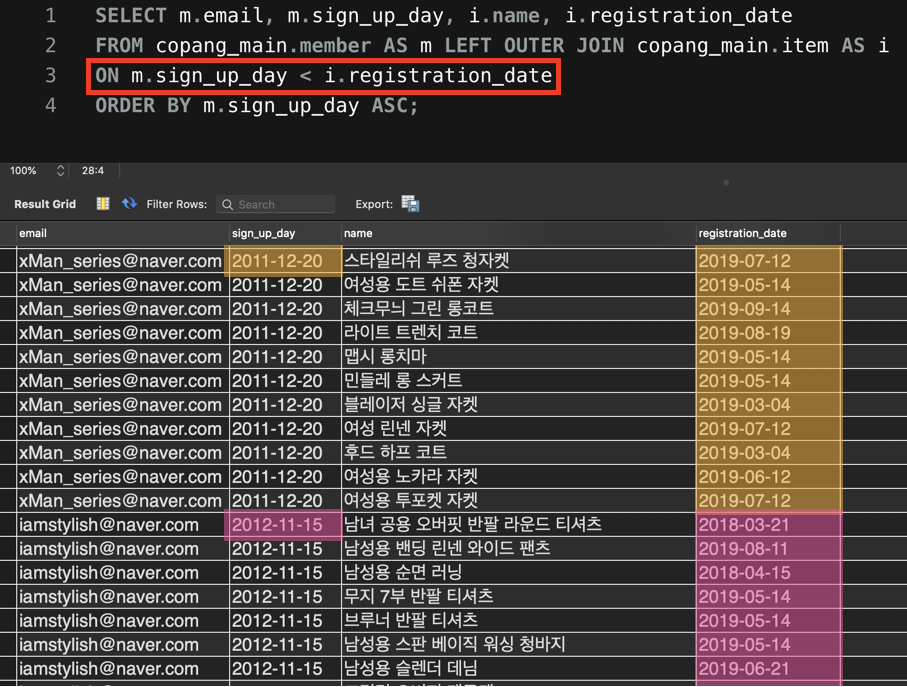
이라고만 써도, 위와 같은 결과를 볼 수 있습니다.

**5. Non-Equi 조인**

자, 마지막으로 이때까지 우리가 살펴봤던 조인과는 전혀 다른 성격의 조인을 살펴보겠습니다. 이때까지 우리는 조인 조건을 설정할 때 두 컬럼의 값이 같은지를 기준으로 했습니다. **즉, 조인 조건에 항상 등호(=)를 사용해왔죠.**

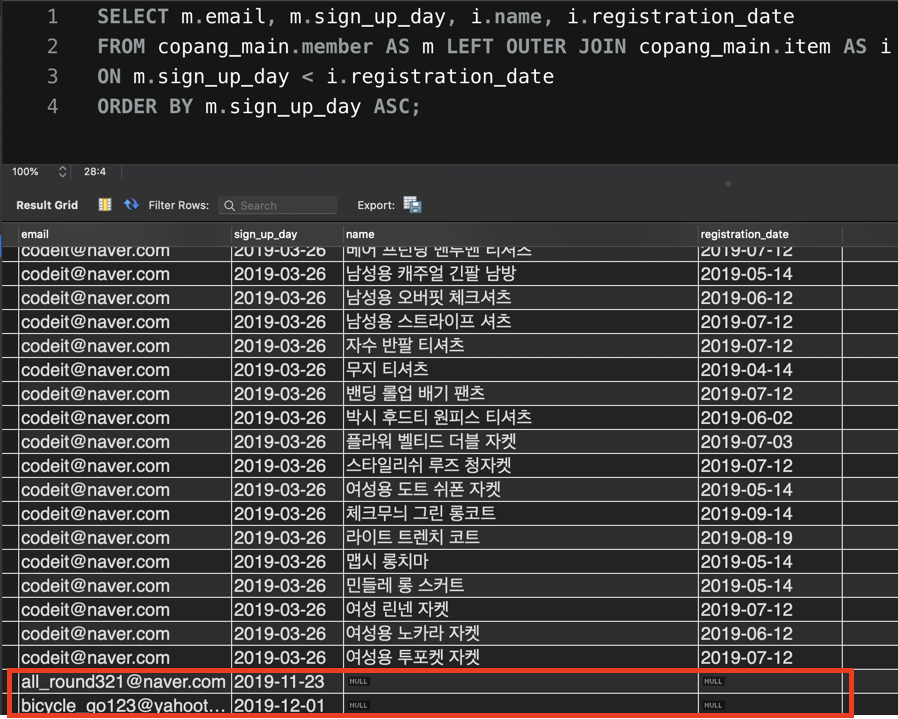
그런 조인들은 **Equi 조인**이라고 합니다. Equi는 Equality Condition의 줄임말로 동등 조건을 의미합니다. 이때까지 우리가 해온 조인은 모두 동등 조건을 판단하는 Equi 조인이었습니다.

하지만 동등 조건이 아닌 다른 종류의 조건을 사용해서 조인을 할 수도 있는데요. 이런 조인을 **Non-Equi 조인**이라고 합니다. 바로 그 예를 보여드릴게요.



지금 저는 member 테이블과 item 테이블을 LEFT OUTER JOIN 했습니다. 그런데 ON 뒤의 조인 조건을 보니 뭔가 좀 이상하죠? **등호가 아니라 부등호(<)가 들어있는데요.**

실행결과를 보면 member 테이블의 sign\_up\_day(회원의 사이트 가입일)보다 더 이후인 registration\_date(상품이 사이트에 등록된 날)을 가진 item들이 연결되었습니다. 이 결과를 보면, 특정 회원이 가입한 이후에 사이트에 올라온 상품들이 무엇인지 확인할 수 있는데요. 좀더 스크롤을 내려볼게요.



지금 가장 최근에 가입한 두 회원 이후로는 새롭게 사이트에 올라온 상품들은 없는 상태라는 걸 알 수 있습니다. 2019년 11월 23일 이후로는 사이트에 새롭게 올라온 상품들이 없네요.

**Non-Equi 조인**이 뭔지 잘 이해되시나요? 사실 Non-Equi 조인은 Equi 조인만큼 보편적으로 사용되지는 않지만 방금 본 것처럼 특정 조건에서는 충분히 유용하게 사용될 수 있는 조인이기 때문에 알아두시는 게 좋습니다. 그리고 Non-Equi 조인에서는 방금 본 부등호 말고도 다양한 조건 표현식이 사용될 수 있는데요. 이 부분은 Non-Equi 조인이라고 검색해서 직접 찾아보시길 추천합니다.

이제 Non-Equi 조인이라는 게 있다는 걸 알게 됐으니까, 나중에 ON절에서 등호(=)가 없는 조건 표현식을 보더라도 당황하지 마세요!