1. **조인(JOIN)**

**1.1 조인?**

관계형 DB 구조에서 테이블은 중복 데이터를 최소화하고 데이터 무결성을 보장하도록 설계되고 만들어진다

개별 테이블에는 가급적 중복성을 피해서 꼭 필요한 데이터만 저장하게 된다. 따라서 테이블을 하나씩 떼어놓고 본다면 정보의 가치가 떨어진다.

사용자에게 의미있는 데이터를 제공하기 위해서 각각의 분리되어 있는 연관성 있는 데이터들을 연결하거나 조합해야 하는데 이를 조인이라고 한다.

ex) Employees 테이블 : 사용자 정보를 담고 있는 테이블

Departments 테이블 : 부서 정보를 담고 있는 테이블

* 두 테이블은 서로 연결지을 때 의미있는 정보가 될 수 있다.

ex1) JOIN을 사용하지 않은 결과물의 단점

Job\_ID, Department\_id가 무엇인가?

|  |
| --- |
| **SELECT** first\_name, last\_name, email, phone\_number, hire\_date,  salary, job\_id, department\_id  **FROM** employees; |

ex2) JOIN을 사용한 결과물

|  |
| --- |
| **SELECT** employees.first\_name, employees.last\_name, employees.email,  departments.department\_name  **FROM** employees, departments  **WHERE** employees.department\_id = departments.department\_id; |

\*) Join 시에는 조회 대상 테이블이 두 개 이상으므로 컬럼을 명시할 때 항상 ‘테이블명.컬럼명’

형식을 따라야 한다.

ex3) 에러

|  |
| --- |
| **SELECT** employees.first\_name, employees.last\_name, employees.email,  departments.department\_name,  department\_id  **FROM** employees, departments  **WHERE** employees.department\_id = departments.department\_id;  *SQL Error : ORA-00918 : 열의 정의가 애매합니다.* |

ex4) 가독성을 위한 문법 : 쿼리문의 단순화(별칭 사용)

|  |
| --- |
| **SELECT** emp.first\_name, emp.last\_name, emp.email,  dep.department\_id, dep.department\_name  **FROM** employees emp, departments dep  **WHERE** emp.department\_id = dep.department\_id; |

ex) 두 개 이상 테이블 Join

|  |
| --- |
| **SELECT** emp.first\_name, emp.last\_name, emp.email,  emp.department\_id, dep.department\_name,  emp.job\_id, job.job\_title  **FROM** employees emp,  departments dep,  jobs job  **WHERE** emp.department\_id = dep.department\_id  **AND** emp.job\_id = job.job\_id; |

\*) AND : 모든 조건이 모두 참일 경우에 조건이 만족된다.

OR : 어느 한 가지 조건만 참이더라도 조건을 만족한다.

🡺 지금까지 조인은 가장 일반적인 조인 방법으로 이를 내부조인이라한다.

**1.2 조인 종류**

* **Inner Join(내부 조인)** : 가장 일반적인 형태의 조인

외부 조인 이외의 조인

일반적으로 조인이라고 하면 내부 조인을 말하는 것임

테이블 간의 공통 컬럼을 사용하여 이 컬럼값이 같은 데이터들을 연결한 조인 방법임

내부 조인의 유형별 분류

두 개 이상의 테이블 조인 / 일반조건과 조인조건을 포함한 조인 / Cartesian Product

셀프 조인 / 기타 조인들(안티 조인 / 세미 조인)

* **Outer Join(외부 조인)** : 내부조인을 확장한 개념

두 테이블이 내부조인으로 연결되었을 때 조인의 결과는 조인조건

에서 명시된 컬럼값이 두 테이블 모두 같은 값을 가진 로우들만

결과로 추출된다.

외부조인은 어느 한 테이블의 공통 컬럼값이 없더라도(컬럼값이 NULL)

해당 로우들이 조회결과에 포함된다.

* **ANSI 내부 조인 / 크로스 조인 / ANSI 외부 조인**

**내부 조인**

1) 두 개 이상의 테이블 조인

\*) 조인은 항상 기준이 되는 테이블이 존재한다.

왜 조인을 하였을까? 사원의 직책명과 부서명을 알기 위함이다.

employees 테이블 : 각 사원들의 직책명(department\_id),

부서명(job\_id)의 접근 ID들이 있다.

구체적인 정보는 해당 테이블에 있다.

결국 기준테이블은 employees이 된다.

ex1)

|  |
| --- |
| **SELECT** emp.first\_name, emp.last\_name, emp.email,  emp.department\_id, dep.department\_name,  emp.job\_id, job.job\_title  **FROM** employees emp,  departments dep,  jobs job  **WHERE** emp.department\_id = dep.department\_id  **AND** emp.job\_id = job.job\_id; |

ex2) 위의 SQL 문장에 또 다른 조인조건 추가 :사원이 소속된 부서가 어느 도시에 있나?

|  |
| --- |
| **SELECT** emp.first\_name, emp.last\_name, emp.email,  emp.department\_id, dep.department\_name,  emp.job\_id, job.job\_title,  loc.city  **FROM** employees emp,  departments dep,  jobs job,  locations loc  **WHERE** emp.department\_id = dep.department\_id  **AND** emp.job\_id = job.job\_id  **AND** dep.location\_id = loc.location\_id; |

\*) EMPLOYEES 테이블과 JOB, DEPARTMENTS 테이블을 조인한 후 그 결과를 가지고 다시

LOCATIONS 테이블과 조인하여 부서가 위치한 도시를 보여준다.

2) 일반조건과 조인조건을 포함한 조인

Where 절에서는 조인조건 이외의 다른 조건들도 명시할 수 있다.

만약, 켈리포니아 주에서 근무하는 사원들의 명단 추출을 요구한다면?

|  |
| --- |
| **SELECT** emp.first\_name, emp.last\_name, emp.email,  dep.department\_name, job.job\_title,  loc.city  **FROM** employees emp,  departments dep,  jobs job,  locations loc  **WHERE** emp.department\_id = dep.department\_id  **AND** emp.job\_id = job.job\_id  **AND** dep.location\_id = loc.location\_id  **AND** loc.state\_province = '캘리포니아'; |

\*) 조인조건 이외의 조건이 WHERE 절에 명시되면 보통 가장 먼저 일반 조건을 처리한 뒤 조인조건을 처

리한다.

1. LOCATIONS 테이블에서 STATE\_PROVINCE 값이 캘리포니아인 로우를 찾는다.
2. 찾은 LOCATIONS\_ID 값과 같은 값을 가진 데이터를 DEPARTMENTS 테이블에서 찾아 조인을 한다.
3. (2)의 결과와 EMPLOYESS 테이블을 비교하여 (DEPARTMENT\_ID 값이 같은 것) 조인한다.
4. (3)의 결과와 JOBS 테이블을 비교하여 조인하여 최종 쿼리결과를 얻는다.

🡺 기준 테이블은 LOCATIONS가 된다.

즉 LOCATIONS 테이블에서 STATE\_PROVINCE값이 캘리포니아인 로우수와 전체쿼리 결과

로우수가 같게 된다는 의미이다.

3) Cartesian Product :

WHERE 절에 조인조건을 넣지 않을 경우?

|  |
| --- |
| **SELECT** a.employee\_id,  a.first\_name,  a.email,  b.department\_name  **FROM** employees a, departments b;  *// 2889건 출력(107\*27건)* |
| **SELECT** a.employee\_id,  a.first\_name,  a.email,  a.department\_id,  b.department\_name  **FROM** employees a, departments b;  *// 2889건 출력(107\*27건)* |
| **SELECT** COUNT(\*)  **FROM** employees;  *//107건* |
| **SELECT** COUNT(\*)  **FROM** departments;  *//27건* |

\*) 카타시안 프로덕트(Cartesian Product) : 가능한 모든 조합의 수를 리턴함

무의미한 쿼리문임

4) 셀프 조인

자기 자신과 조인을 맺는 것을 말함

|  |
| --- |
| **SELECT** a.employee\_id, b.first\_name, b.last\_name  **FROM** employees a, employees b  **WHERE** a.employee\_id = b.employee\_id  **AND** rownum < 5;  // 4건 출력 |

\*) 반드시 별칭을 써야 함

\*) 왜 셀프 조인이 필요할까?

employee테이블에는 manager\_id라는 컬럼이 있다.

이 컬럼의 용도는 해당 사원의 매니져가 누구인지를 나타낸다.

그럼 매니져의 이름이나 다른 정보는 어디에 있을까?

모든 사원의 정보는 employee 테이블에 있다.

|  |
| --- |
| -- 사원번호 / 이름 / 메니저 번호 / 메니져 이름  **SELECT** emp.employee\_id,  emp.first\_name,  man.employee\_id manager\_id,  man.first\_name manager  **FROM** employees emp, employees man  **WHERE** emp.manager\_id = man.employee\_id ; |

|  |
| --- |
| -- 메니져 ID가 없는 사람을 확인할 수 있다.  **SELECT** employee\_id, first\_name  **FROM** employees  **WHERE** manager\_id IS NULL; |

5) 기타 조인들(안티 조인)

- 위의 조인은 조인조건에서 모두 '=' 연산자(equijoin)를 사용했다.

- 동등조인과 달리 '='연산자를 사용하지 않는 조인들도 있다.

- 안티조인은 말 그대로 동등조인과는 반대되는 개념의 조인을 말한다.

연산자로 NOT IN을 사용한다.

- 정상 문장

location\_id값이 3200인 , 즉 봄베이에 위치한 부서 이외의 모든 부서에 속한

사원 추출하는 SQL문

|  |
| --- |
| **SELECT** emp.employee\_id, emp.first\_name  **FROM** employees emp  **WHERE** emp.department\_id NOT IN  ( SELECT dep.department\_id  FROM departments dep  WHERE dep.location\_id = 3200 ); |

\*) 안티조인이란 조인조건에서 NOT IN 연산자를 사용하는 조인을 말한다.

6) 기타 조인들(세미 조인)

- NOT IN 대신 EXISTS를 사용한다.

|  |
| --- |
| - 월 급여가 10,000달러 이상인 사원이 속한 부서를 추출  **SELECT** dep.\*  **FROM** departments dep  **WHERE** **EXISTS** ( SELECT 1  FROM employees emp  WHERE emp.department\_id = dep.department\_id  AND emp.salary > 10000 )  **ORDER BY** dep.department\_id; |

\*) NOT IN 으로 작성 가능

**외부조인**

1) Outer Join(외부 조인)

내부조인을 확장한 개념

두 테이블이 내부조인으로 연결되었을 때 조인의 결과는 조인조건에서 명시된 컬럼값이 두 테이

블 모두 같은 값을 가진 로우 들만 결과로 추출된다.

외부조인은 어느 한 테이블의 공통 컬럼값이 없더라도(컬럼값이 NULL) 해당 로우들이 조회결과

에 포함된다.

ex) 만약 department\_id 가 NULL인 사람이 있다면 결과에서 제외된다.

|  |
| --- |
| **SELECT** emp.employee\_id,  emp.first\_name,  emp.department\_id,  dep.department\_name  **FROM** employees emp, departments dep  **WHERE** emp.department\_id = dep.department\_id; |

\*) 모든 사원의 이름과 소속부서를 보고자 한다.

하지만 소속부서가 없는 사람은 결과에서 제외되게 된다.

이러한 내부조인의 문제를 외부조인이 해결하게 된다.

ex) 만약 department\_id 가 NULL인 사람이 있어도 결과에 포함된다.

즉, 외부조인은 조인조건을 만족하는 데이터를 추출하되 어느 한 테이블에만

데이터가 존재한다면 다른 테이블에 해당 데이터가 존재하지 않더라도 조인결과에

포함시키는 조인을 말한다.

- 데이터가 존재하지 않는 테이블의 조인조건에 (+)를 붙여야 한다.

|  |
| --- |
| **SELECT** emp.employee\_id,  emp.first\_name,  emp.department\_id,  dep.department\_name  **FROM** employees emp, departments dep  **WHERE** emp.department\_id = dep.department\_id (+)  **ORDER BY** emp.department\_id; |

\*) (+) 해당 테이블에 데이터가 존재하지 않더라도 무시하고 조인에 참여하라.

- 외부 조인의 제약사항

위 쿼리에서 부서명뿐만 아니라 부서가 위치한 도시명도 알고 싶을 경우

부서정보는 LOCATONS 테이블에 있으므로 다음과 같이 이 테이블도 추가한다.

|  |
| --- |
| **SELECT** emp.employee\_id,  emp.first\_name,  emp.department\_id,  dep.department\_name  **FROM** employees emp, departments dep, locations loc  **WHERE** emp.department\_id = dep.department\_id(+)  **AND** dep.location\_id = loc.location\_id  **ORDER BY** emp.department\_id; |

\*) emp.department\_id = dep.department\_id(+) 외부조언

dep.location\_id = loc.location\_id 내부조인

🡺 LOCATIONS 테이블을 조회조건에 포함시켰더니 다시 NULL값이 배제되었다.

내부 조인 적용과정에서 NULL값이 다시 배제되는 문제가 발생한다.

그러면 어디에다가 외부조인 처리를 해야 하나?

ex) 위 코드 개선

여러 개의 테이블이 조인될 경우 기준 테이블을 중심으로 맨 먼저 기준테이블과

조인조건이 걸린 테이블과의 조인을 먼저 처리하고 그 결과를 가지고 다시 나머지 테이블과

의 조인을 처리한다.

즉, 첫번째 조인조건을 처리한 뒤 그 결과를 가지고 다시 LOCATIONS 테이블과 조인을 처리

한다.

|  |
| --- |
| **SELECT** emp.employee\_id,  emp.first\_name,  dep.department\_name,  loc.city  **FROM** employees emp,  departments dep,  locations loc  **WHERE** emp.department\_id = dep.department\_id(+)  **AND** dep.location\_id = loc.location\_id(+)  **ORDER BY** emp.department\_id; |

규칙1) (+)는 where 절에서만 사용이 가능하다.--------------

규칙2) 테이블간 외부조인 조건이 한 개 이상일 경우, 모든 외부조인 조건에 (+)를 붙여야

외부조인이 성립된다.

|  |
| --- |
| // null제외코드  **SELECT** e.employee\_id, e.first\_name,  j.start\_date, j.end\_date, j.job\_id, j.department\_id old\_dept  **FROM** employees e, job\_history j  **WHERE** e.employee\_id = j.employee\_id(+)  **AND** j.department\_id = 80  **ORDER BY** j.employee\_id; |
| // 개선 코드  **SELECT** e.employee\_id, e.first\_name,  j.start\_date, j.end\_date, j.job\_id, j.department\_id old\_dept  **FROM** employees e, job\_history j  **WHERE** e.employee\_id = j.employee\_id(+)  **AND** j.department\_id(+) = 80  **ORDER BY** j.employee\_id; |

규칙3) 테이블 자신에 (+) 를 붙일 수 없다.

|  |
| --- |
| // 에러  **SELECT** employee\_id, manager\_id  **FROM** employees  **WHERE** manager\_id(+) = employee\_id;  *Error report:*  *SQL Error : ORA-01416 : 두 개의 테이블을 outer-join 할 수 없습니다.* |
| // 개선 : 셀프 조인을 활용한다.  **SELECT** e1.employee\_id, e1.manager\_id, e2.employee\_id  **FROM** employees e1, employees e2  **WHERE** e1.manager\_id(+) = e2.employee\_id; |

규칙4) (+) 는 오직 컬럼에만 붙일 수 있으며 or 연산자와는 같이 사용할 수 없다.

|  |
| --- |
| // ERROR  **SELECT** e.employee\_id, e.first\_name, e.department\_id current\_dept,  j.start\_date, j.end\_date, j.job\_id, j.department\_id old\_dept  **FROM** employees e,  job\_history j  **WHERE** e.employee\_id = j.employee\_id(+)  **OR** e.department\_id = j.department\_id  **ORDER BY** j.employee\_id;  *Error report:*  *SQL Error : ORA-01719 : 포괄 조인 운영(+) 는 OR 또는 IN의 연산수를 허용하지 않습니다.* |

규칙5) 외부조인시에 대상 테이블은 오직 한 개의 테이블과만 외부조인이 가능하다.

|  |
| --- |
| **SELECT** e.employee\_id, e.first\_name, e.department\_id current\_dept,  j.start\_date, j.end\_date, j.job\_id, j.department\_id old\_dept, d.department\_name  **FROM** employees e, job\_history j, departments d  **WHERE** e.employee\_id = j.employee\_id(+)  **AND** e.department\_id = j.department\_id(+)  **AND** d.department\_id = j.department\_id(+)  **ORDER BY** j.employee\_id;  *Error report:*  *SQL Error : ORA-01417 : 하나의 테이블은 하나의 다른 테이블과 포괄 조인할 수 있습니다.* |

\*) SELECT 문장에서 대상 테이블이 A, B, C 가 있을 경우,

여기서 A와 B나 B나 C 혹은 A와 C 테이블 간에는 외부조인을 맺을 수 있지만

A와 B, 그리고 B와 C 테이블에 동시에 외부조인을 맺을 수 없다.

규칙6) 조인조건의 오직 한 쪽에만 (+) 를 붙일 수 있다.

|  |
| --- |
| **SELECT** dep.department\_id, dep.department\_name,  emp.employee\_id, emp.first\_name  **FROM** employees emp,  departments dep  **WHERE** dep.manager\_id(+) = emp.employee\_id(+);  *Error report:*  *SQL Error : ORA-01418 : outer-join 된 테이블은 1개만 지정할 수 있습니다.* |

**ANSI 조인 : SQL 표준 쿼리문**

1) ANSI 내부 조인

==> 기존 전통적인 내부 조인문장

SELECT emp.employee\_id, emp.first\_name, dep.department\_name

FROM employees emp, departments dep

WHERE emp.department\_id = dep.department\_id;

==> ANSI 내부 조인 문장

SELECT emp.employee\_id, emp.first\_name, dep.department\_name

FROM employees emp INNER JOIN departments dep

ON emp.department\_id = dep.department\_id;

==> ANSI 내부 조인 문장

ON대신 USING을 사용하여 컬럼 이름을 한번만 기술하면 된다.

SELECT emp.employee\_id, emp.first\_name, dep.department\_name

FROM employees emp INNER JOIN departments dep

USING ( department\_id );

// 문법적 에러 : USING을 사용할 때는 컬럼명만 기술해야 한다.

SELECT emp.employee\_id, emp.first\_name, dep.department\_name,

dep.department\_id

FROM employees emp INNER JOIN departments dep

**USING** ( department\_id );

// => 위의 에러 수정

SELECT emp.employee\_id, emp.first\_name, dep.department\_name,

department\_id

FROM employees emp INNER JOIN departments dep

**USING** ( department\_id );

//=> ON의 사용

emp.employee\_id = dep.manager\_id;

join하는 컬럼명이 다르므로 USING사용이 불가능 하다.

SELECT emp.employee\_id, emp.first\_name, dep.department\_name

FROM employees emp INNER JOIN departments dep

**ON** emp.employee\_id = dep.manager\_id;

2) 크로스 조인

카타시안 프로덕트를 ANSI에서는 크로스 조인이라고 한다.

SELECT \*

FROM employees CROSS JOIN departments;

3) ANSI 외부조인

--> 전통적인 방법

SELECT Jhis.employee\_id, emp.first\_name

FROM employees emp, job\_history jhis

WHERE emp.employee\_id = jhis.employee\_id(+)

ORDER BY jhis.employee\_id;

--> ANSI 변경 문법

SELECT Jhis.employee\_id, emp.first\_name

FROM employees emp LEFT OUTER JOIN job\_history jhis

ON emp.employee\_id = jhis.employee\_id

ORDER BY jhis.employee\_id;

--> ANSI변경 문법

SELECT Jhis.employee\_id, emp.first\_name

FROM job\_history jhis RIGHT OUTER JOIN employees emp

ON emp.employee\_id = jhis.employee\_id

ORDER BY jhis.employee\_id;