Relational Algebra

Dr. E.F.Codd의 RDBMS 논문(1970)을 기초로 한다.

- 일반 집합 연산자

연산자	SQL	설명
UNION	UNION UNION ALL	합집합 (중복 제거) 합집합 (중복 허용)
INTERSECTION	INTERSECT	교집합 (공통 요소 추출)
DIFFERENCE	EXCEPT (Oracle : MINUS)	차집합 (앞SQL기준으로 - 뒤SQL)
PRODUCT	CROSS JOIN	곱집합 (JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합, M*N) (= CARTESIAN PRODUCT)

- 순수 관계 연산자

DBMS를 구현하기 위해 새롭게 만들어진 연산자

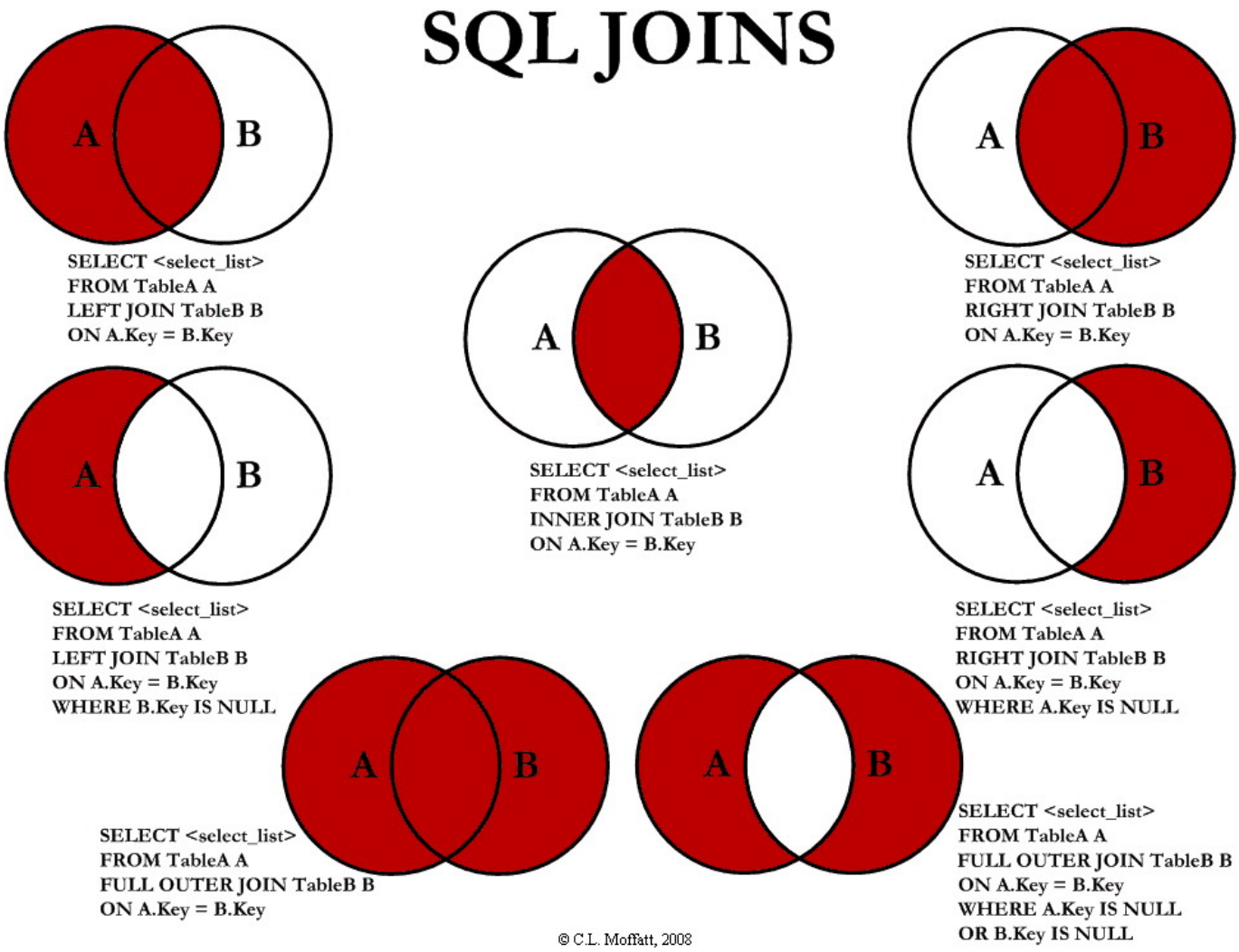
연산자	SQL	비고
SELECT	WHERE	
PROJECT	SELECT	
(NATURAL) JOIN	NATURAL JOIN INNER JOIN OUTER JOIN USING 조건절 ON 조건절 ...	
DIVIDE		현재 사용하지 않음

Data Manipulation Language, SQL, Database

JOIN

- 두 개 이상의 Table들을 연결/결합하여 데이터를 출력하는 것
- 단 두개의 집합 간에만 JOIN 연산이 일어난다. 나머지 집합에 대해서는 새 조인 집합과 순차적으로 연산된다.
- 같은 이름의 중복 칼럼의 경우, **테이블명/Alias가 필수 조건**이다.

n개의 테이블에서 필요 데이터 조회시 필요한 JOIN 조건은 **대상(연산 기준) 테이블** 개수 하나를 뺀 **n-1개 이상**이 필요하다. (= 피연산 테이블 개수가 n-1)



기본 개념

EQUI JOIN

- 두 개의 Table 간에 칼럼 값들이 서로 정확하게 일치하는 경우
  - 주로 PK-FK 연간관계
  - 논리적으로 같은 값이 존재
- WHERE 절에서 "=" 연산자를 사용해서 표현

```
SELECT 테이블1.칼럼명, 테이블2.칼럼명, ... -- 유일성 보장을 위해 테이블명을 명시하는 습관을 들이자!
FROM 테이블1, 테이블2
WHERE 테이블1.칼럼명1 = 테이블2.칼럼명2;


/*ANSI/SQL 표준 방식 (INNER JOIN 명시)*/
SELECT 테이블1.칼럼명, 테이블2.칼럼명, ... -- 유일성 보장을 위해 테이블명을 명시하는 습관을 들이자!
FROM 테이블1 INNER JOIN 테이블2
ON 테이블1.칼럼명1 = 테이블2.칼럼명2;
```

- 예시

```
SELECT P.PLAYER_NAME, T.TEAM_ID, STADIUM_NAME
FROM PLAYER P, TEAM T, STADIUM S
WHERE P.TEAM_ID = T.TEAM_ID
      AND T.STADIUM_ID = S.STADIUM_ID
ORDER BY PLAYER_NAME;

SELECT P.PLAYER_NAME, T.TEAM_ID, STADIUM_NAME
FROM PLAYER P INNER JOIN TEAM T
ON P.TEAM_ID = T.TEAM_ID
   INNER JOIN STADIUM S
   ON T.STADIUM_ID = S.STADIUM_ID
ORDER BY PLAYER_NAME;
```

Non EQUI JOIN

- 두 개의 Table 간에 칼럼 값들이 서로 정확하게 일치하지 않는 경우
- WHERE 절에서 "="가 아닌 다른 연산자를 사용한다.
    - BETWEEN ~ AND ~, >, >=, <, ⇐, ...
      ☞ 일치하는 것이 "점"의 느낌이라면, 이건 마치 "선(범위)"의 느낌.

FROM 절 JOIN 형태

SQL Server의 경우 ON 조건절만 지원, NATURAL JOIN과 USING 조건절은 미지원,

INNER JOIN

- WHERE 절에서 사용하던 JOIN의 Default Option, 생략 가능하나 CROSS/OUTER JOIN과 같이 사용 불가.
- JOIN 조건에서 동일한 값이 있는 행만 반환.
- WHERE 절에서 사용하던 JOIN 조건을 FROM 절에서 정의하겠다는 표시, USING/ON 조건절을 필수적으로 사용해야 한다.
- "*" 사용시 칼럼들이 테이블 순서대로 출력된다. (같은 칼럼명이 있어도 중복 출력)
    - ☞ 단순히 Table 두개를 나란히 붙임. 중복을 허용한 합집합의 느낌.

```
/* WHERE절 JOIN 조건 */
SELECT 테이블.칼럼명, ...
FROM 테이블1[ [AS] 별칭], 테이블2[ [AS] 별칭]
WHERE 테이블1.칼럼명 = 테이블2.칼럼명;
```

```
/* FROM 절 JOIN 조건 */
SELECT 테이블.칼럼명, ...
FROM 테이블1[ [AS] 별칭] [INNER] JOIN 테이블2[ [AS] 별칭]
ON 테이블1.칼럼명 = 테이블2.칼럼명;
```

NATURAL JOIN

- INNER JOIN의 하위 개념 (= NATURAL (INNER) JOIN)
- 두 Table간의 동일한 이름을 갖는 모든 칼럼들에 대해 EQUI JOIN 수행.
- NATURAL JOIN에 사용되는 칼럼들은 ①**같은 Type**이어야 하고, ②**접두사(Alias/테이블명)를 붙일 수 없음**.
- "*" 사용시 NATURAL JOIN의 기준이 되는 칼럼들이 **중복없이 먼저** 출력된다.
    - ☞ 겹치는 부분은 한번만 사용하는 합집합의 느낌.
- WHERE 절에 JOIN 조건을 추가할 수 없다.
    - ☞ 겹치는 부분 딱 한번 나오니까 더 이상 조건 필요 없다! 당연한 소리!
- SQL Server는 미지원

```
SELECT 칼럼명, ...
FROM 테이블1 NATURAL [INNER] JOIN 테이블2
```

OUTER JOIN

- JOIN 조건에서 동일한 값이 없는 행도 반환, NULL로 채운다.
- JOIN 조건을 FROM 절에서 정의하겠다는 표시로 **USING/ON 조건절을 필수적**으로 사용해야 한다.

LEFT OUTER JOIN

- 좌측 테이블 기준 (→)
- 우측 테이블 JOIN 칼럼에 같은 값이 없는 경우 NULL로 채운다.

```
SELECT 테이블명.칼럼명, ...
FROM 테이블명1[ [AS] 별칭] LEFT [OUTER] JOIN 테이블명2[ [AS] 별칭]
ON 테이블명1.칼럼명 = 테이블명2.칼럼명;
```

RIGHT OUTER JOIN

- 우측 테이블 기준 (←)
- 좌측 테이블 JOIN 칼럼에 같은 값이 없는 경우 NULL로 채운다.

```
SELECT 테이블명.칼럼명, ...
```

```
FROM 테이블명1[ [AS] 별칭] RIGHT [OUTER] JOIN 테이블명2[ [AS] 별칭]
ON 테이블명1.칼럼명 = 테이블명2.칼럼명;
```

FULL OUTER JOIN

- 좌/우측 모든 테이블 기준
- 좌/우측 테이블 JOIN 칼럼에 같은 값이 없는 경우 NULL로 채운다.
- UNION 기능과 같으므로 중복없는 합집합 처리 결과다.

```
SELECT 테이블명.칼럼명,...
FROM 테이블명1[ [AS] 별칭] FULL [OUTER] JOIN 테이블명2[ [AS] 별칭]
ON 테이블명1.칼럼명 = 테이블명2.칼럼명;

-- "LEFT/RIGHT JOIN 후 UNION"하면 아래 결과와 같다

-- SELECT 테이블명.칼럼명,...
-- FROM 테이블명1[ [AS] 별칭] LEFT [OUTER] JOIN 테이블명2[ [AS] 별칭]
-- ON 테이블명1.칼럼명 = 테이블명2.칼럼명;
-- UNION
-- SELECT 테이블명.칼럼명,...
-- FROM 테이블명1[ [AS] 별칭] RIGHT [OUTER] JOIN 테이블명2[ [AS] 별칭]
-- ON 테이블명1.칼럼명 = 테이블명2.칼럼명;
```

조건절

ON 조건절

- 과거에 WHERE 절에서 JOIN 조건과 데이터 검증 조건이 같이 사용되어 용도가 불분명한 경우가 발생할 수 있었음. ("="의 혼동)
  - ☞ FROM 절의 ON 조건절로 분리하여 이해 높임.
    - ☞ JOIN 서술부(ON 조건절)와 비서술부(WHERE 조건절)를 분리하여 이해가 쉽다.
    - ☞ (NATURAL JOIN이나 USING 조건절처럼) 칼럼명이 서로 다르더라도 JOIN 조건으로 사용할 수 있다.
- FROM 절에 테이블이 많이 사용될 경우 가독성이 떨어지는 단점이 있다.

- 예시1

WHERE, ORDER BY 혼용

```
SELECT T.TEAM_NAME, T.TEAM_ID, S.STADIUM_NAME
FROM TEAM T JOIN STADIUM S
ON T.TEAM_ID = S.HOMETEAM_ID
ORDER BY TEAM_ID;
```

```
SELECT T.TEAM_NAME, T.TEAM_ID, S.STADIUM_NAME
FROM TEAM T, STADIUM S
WHERE T.TEAM_ID = S.HOMETEAM_ID
ORDER BY TEAM_ID;
```

- 예시2

다중 Table JOIN

```
SELECT E.EMPNO, D.DEPTNO, D.DNAME, T.DNAME NEW_DNAME
FROM EMP E JOIN DEPT D
ON (E.DEPTNO = D.DEPTNO)
   JOIN DEPT_TEMP T
   ON E.DEPTNO = T.DEPTNO;

SELECT E.EMPNO, D.DEPTNO, D.DNAME, T.DNAME NEW_DNAME
FROM EMP E, DEPT D, DEPT_TEMP T
WHERE (E.DEPTNO = D.DEPTNO)
      AND (E.DEPTNO = T.DEPTNO);
```

USING 조건절

- 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI JOIN
- "*" 사용시 USING 조건절의 기준이 되는 칼럼 먼저 출력.
- JOIN 칼럼에 접두사(Alias/테이블명) 붙일 수 없음.
- SQL Server 미지원

```
SELECT 칼럼명, ...
FROM 테이블1 [INNER] JOIN 테이블2
USING (칼럼명[, 칼럼명]);
```

CROSS JOIN

- Table 간 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합 (M*N)
- 일반 집합 연산자의 PRODUCT의 개념. (= CARTESIAN/CROSS PRODUCT)
- WHERE 절에 JOIN 조건 사용 = INNER JOIN (☞ 의미 없다.)

```
SELECT 테이블명.칼럼명, ...
FROM 테이블1[ [AS] 별칭] CROSS JOIN 테이블2[ [AS] 별칭];
```

Set Operator

집합 연산자

- 두 개 이상의 Table에서 JOIN을 이용하지 않고 연관된 데이터를 조회하는 방법
- 여러 개의 질의 결과를 연결하여 하나로 결합하는 방식

- 제약조건
  - SELECT 절의 칼럼 수 동일
  - SELECT 절의 동일 위치에 존재하는 칼럼의 Data Type 상호 호환 가능(☞ 반드시 동일 Data Type일 필요X)

집합 연산자	설명	비고
UNION	합집합 (중복 제거)	WHERE 절에 “IN”(같은 칼럼일때만), “OR” 연산자로 변환 가능
UNION ALL	합집합 (중복 허용)	“NOT EXISTS”, “NOT IN” Sub Query로 변환 가능
INTERSECT	교집합 (공통 요소 추출)	“EXISTS”, “IN” Sub Query로 변환 가능
EXCEPT (Oracle : MINUS)	차집합 (앞SQL - (교집합))	

```
SELECT ~ [HAVING ~]
집합_연산자
SELECT ~ [HAVING ~]
[ORDER BY ~]
```

Hierarchical Query

- 계층형 질의 : 계층형 데이터가 존재하는 경우 데이터를 조회하기 위한 질의
  - 상하 관계를 표시해서 출력하므로 가독성 향상
    - ☞ 동일 테이블에 계층적으로 상위와 하위 데이터가 포함된 데이터

```
SELECT 칼럼명, …
FROM 테이블명, …
WHERE 조건, …
START WITH 조건
CONNECT BY [NOCYCLE] 조건, …
[ORDER SIBLINGS BY 칼럼명, …]
```

- 실행 순서
  1. START WITH 절 : 계층 구조 전개의 시작 위치 지정 구문. 즉, 루트 데이터를 지정. (Access)
     - PRIOR : 해당 키워드가 설정되어 있는 칼럼에서 바로 이전 데이터의 값을 찾는 데 사용. (현재 읽은 칼럼 지정, 대입연산으로 이해)
       - PRIOR 자식 = 부모 ☞ 부모→자식 (순방향)
       - PRIOR 부모 = 자식 ☞ 자식→부모 (역방향)
     - NOCYCLE : 사이클이 발생한 이후의 데이터 전개X

2. CONNECT BY 절 : 자식 데이터 지정 구문. 주어진 조건을 만족해야 함. (JOIN)
3. WHERE : 모든 전개를 수행 후, 지정된 조건을 만족하는 데이터만 추출. (Filtering)
4. ORDER SIBINGS BY : 형제 노드(동일 Level) 사이에서 정렬 수행

Pseudo column

(Oracle 기준)

- LEVEL : 해당 데이터가 몇 번째 단계인 지 의미. (루트 데이터면 1, 그 하위 데이터면 2. Leaf 데이터까지 1씩 증가.)
- CONNECT_BY_ISLEAF : 전개 과정에서 해당 데이터가 리프 데이터면 1, 아니면 0.
- CONNECT_BY_ISCYCLE : 전개 과정에서 자식을 갖는 데, 해당 데이터가 조상으로 존재하면 1, 아니면 0.
    - CYCLE 옵션을 사용했을 때만 사용 가능!

순방향 전개

- 예시1

```
SELECT LEVEL,
       LPAD('~', (LEVEL-1)*4) || EMPNO,
    --왼쪽 기준 "(LEVEL-1)*4"위치에 '~'삽입하고, EMPNO 칼럼을 덛붙인다.
       MGR,
       CONNECT_BY_ISLEAF --Leaf면 1, 아니면 0
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR; --MGR→EMPNO
```

	LEVEL	LPAD('~',4*(LEVEL-1))||EMPNO	MGR	CONNECT_BY_ISLEAF
1	1	7839	(null)	0
2	2	~7566	7839	0
3	3	~7788	7566	0
4	4	~7876	7788	1
5	3	~7902	7566	0
6	4	~7369	7902	1
7	2	~7698	7839	0
8	3	~7499	7698	1
9	3	~7521	7698	1
10	3	~7654	7698	1
11	3	~7844	7698	1
12	3	~7900	7698	1
13	2	~7782	7839	0
14	3	~7934	7782	1

- 예시2

```
SELECT LEVEL,
       LPAD(EMPNO, LEVEL*4, '~'),
    --왼쪽 기준 "LEVEL*4"byte 길이로 출력하되 빈 자리는 '~'로 채운다.
       MGR,
       CONNECT_BY_ISLEAF --Leaf면 1, 아니면 0
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR; --MGR→EMPNO
```

	LEVEL	LPAD(EMPNO,LEVEL*4,'~')	MGR	CONNECT_BY_ISLEAF
1	1	7839	(null)	0
2	2	~~~~7566	7839	0
3	3	~~~~~~~~7788	7566	0
4	4	~~~~~~~~~~~~7876	7788	1
5	3	~~~~~~~~7902	7566	0
6	4	~~~~~~~~~~~~7369	7902	1
7	2	~~~~7698	7839	0
8	3	~~~~~~~~7499	7698	1
9	3	~~~~~~~~7521	7698	1
10	3	~~~~~~~~7654	7698	1
11	3	~~~~~~~~7844	7698	1
12	3	~~~~~~~~7900	7698	1
13	2	~~~~7782	7839	0
14	3	~~~~~~~~7934	7782	1

역방향 전개

```
SELECT LEVEL, LPAD(EMPNO, LEVEL*4, '~'), MGR, CONNECT_BY_ISLEAF
FROM EMP
START WITH EMPNO = '7876'
CONNECT BY EMPNO = PRIOR MGR;
```

	LEVEL	LPAD(EMPNO,LEVEL*4,'~')	MGR	CONNECT_BY_ISLEAF
1	1	7876	7788	0
2	2	~~~~7788	7566	0
3	3	~~~~~~~~7566	7839	0
4	4	~~~~~~~~~~~~7839	(null)	1

Built-in function

(Oracle 기준)

SYS_CONNECT_BY_PATH

루트 데이터로부터 현재 전개할 데이터까지의 경로 표시

```
SYS_CONNECT_BY_PATH(칼럼, 경로분리자)
```

CONNECT_BY_ROOT

현재 전개할 데이터의 루트 데이터 표시.

```
CONNECT_BY_ROOT(칼럼)
```

SELF JOIN

- 동일 Table간의 JOIN
  - ☞ When? 동일 Table에서 세부 분류가 필요할 때!
- 동일 Table을 사용하기 때문에 별칭(Alias) 필수!

```
SELECT 별칭1.칼럼명, 별칭2.칼럼명, ...
FROM 테이블명 [AS] 별칭1, 테이블명 [AS] 별칭2
WHERE 별칭1.칼럼명 = 별칭2.칼럼명
```

From:
https://ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
https://ledyx.xyz/doku.php?id=relational_database:relational_algebra

Last update: **2021/02/07 04:16**