[AI초급] SQLD 자격증 코스 - 챕터 11

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- 여러 테이블에 흩어진 데이터를 연결하여 조회하는 JOIN에 대해 학습합니다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. JOIN

관계형 데이터베이스의 핵심적인 기능 JOIN에 대해 학습합니다.

▼ 1) EQUI JOIN

☑️ JOIN 개요

두 개 이상의 테이블을 연결 또는 결합하여 데이터를 출력하는 것을 JOIN 이라고 부릅니다. 일반적으로 하나의 데이터베이스에 테이블은 1개가 아닌 여러 개로 구성되어 있으며 테이블 간 관계가 있습니다. 그렇기 때문에 일반적으로 사용되는 SQL 문장들의 대부분은 JOIN을 통해 데이터를 조회합니다. 이는 관계형 데이터베이스의 가장 큰 장점이자 핵심적인 기능입니다.

일반적인 경우 행들은 기본키(PK)나 외래키(FK) 값의 관계 의해서 JOIN이 성립됩니다. 하지만 어떤 경우에는 논리적인 값들의 연관만으로도 JOIN이 가능합니다.

FROM 절에 여러 테이블을 나열하여 JOIN 을 할 수 있는데, 이때 JOIN의 발생은 두 테이블만 JOIN이 이루어집니다. 이렇게 2개의 테이블이 JOIN이 완료되면 그다음 테이블과 방금 JOIN된 테이블과의 두 번째 JOIN이 발생합니다. 이렇게 FROM에 작성된 모든 TABLE과의 JOIN이 발생합니다.

☑️ EQUI JOIN

EQUI(등가) JOIN은 두 테이블 간에 칼럼의 값들이 서로 같은 경우 두 데이터를 하나의 데이터로 합치는 JOIN을 말합니다. 이때 주로 사용되는 칼럼이 PRIMARY KEY(PK)와 FOREIGN KEY(FK)입니다. 물론 PK와 FK 말고도 다른 칼럼의 값으로도 EQUI JOIN이 성립합니다.

- 기본 구조
  - JOIN 할 두 테이블을 FROM 절에 나열합니다.
  - JOIN의 조건은 WHERE 절에 `=` 연산자를 사용하여 표현합니다.

- 테이블명과 칼럼명을 같이 명시하는 이유는 칼럼명이 동일한 경우 어떤 테이블의 칼럼인지 명확하게 알 수 없기 때문입니다. 즉, 가독성과 유지보수를 높이기 위해서 같이 명시합니다.
- JOIN의 조건은 FROM 절에 작성된 테이블의 개수에서 하나를 뺀 N - 1개 이상이 필요합니다. CROSS JOIN이라는 예외가 있지만 이는 다음 장에서 언급하도록 하고 기본 구조에 대해서 살펴보겠습니다.

```
SELECT 테이블1.칼럼명, 테이블2.칼럼명, ...
FROM 테이블1, 테이블2
WHERE 테이블1.칼럼명1 = 테이블2.칼럼명2;
```

(1) 사원-부서 EQUI JOIN 사례

- 테이블 명이 길고 SQL의 복잡도가 높아지면 가독성이 떨어지기 때문에 테이블명 대신 ALIAS를 주로 사용합니다.
- 테이블에 ALIAS를 설정했다면 반드시 ALIAS 설정된 명칭으로 JOIN을 진행해야 합니다.

▼ HR table

TABLE **EMPLOYEES**

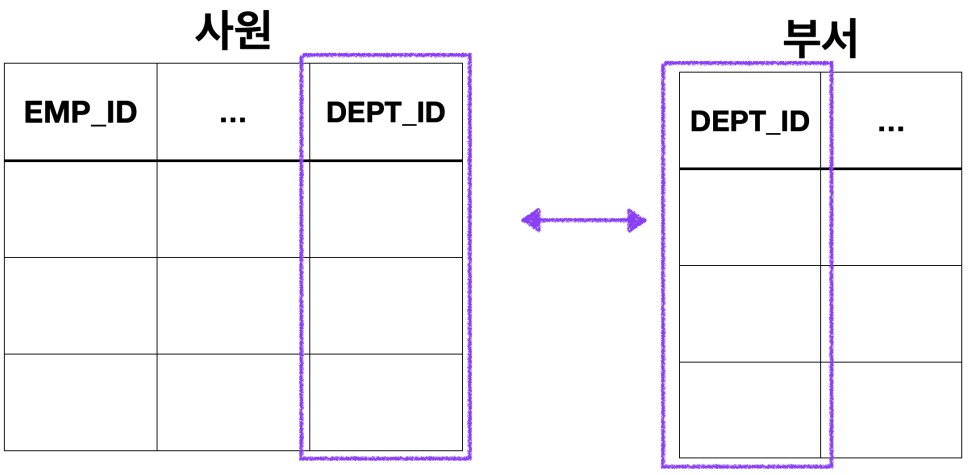
Column	Type	Length	Precision	Scale	Nullable
EMPLOYEE_ID	NUMBER	22	6	0	NO
FIRST_NAME	VARCHAR2	20			YES
LAST_NAME	VARCHAR2	25			NO
EMAIL	VARCHAR2	25			NO
PHONE_NUMBER	VARCHAR2	20			YES
HIRE_DATE	DATE	7			NO
JOB_ID	VARCHAR2	10			NO
SALARY	NUMBER	22	8	2	YES
COMMISSION_PCT	NUMBER	22	2	2	YES
MANAGER_ID	NUMBER	22	6	0	YES
DEPARTMENT_ID	NUMBER	22	4	0	YES

TABLE **DEPARTMENTS**

Column	Type	Length	Precision	Scale	Nullable
DEPARTMENT_ID	NUMBER	22	4	0	NO
DEPARTMENT_NAME	VARCHAR2	30			NO
MANAGER_ID	NUMBER	22	6	0	YES
LOCATION_ID	NUMBER	22	4	0	YES

TABLE **JOBS**

Column	Type	Length	Precision	Scale	Nullable
JOB_ID	VARCHAR2	10			NO
JOB_TITLE	VARCHAR2	35			NO
MIN_SALARY	NUMBER	22	6	0	YES
MAX_SALARY	NUMBER	22	6	0	YES



사원-부서 EQUI JOIN TABLE

EMP_ID	NAME	...	DEPT_ID	...

```
SELECT
  '사원'.employee_id,
  '사원'.last_name,
  '사원'.department_id,
  '부서'.department_name
FROM employees '사원', departments '부서'
WHERE '사원'.department_id = '부서'.department_id;
```

- 실습

▼ 실습 데이터

```
CREATE TABLE PARTICIPANT (
partic_id NUMBER,
nation_id NUMBER,
main_sport_id NUMBER,
first_name VARCHAR2(30),
last_name VARCHAR2(30),
gender VARCHAR2(1),
height NUMBER,
weight NUMBER
);

INSERT INTO PARTICIPANT VALUES (1, 1, 101, 'John', 'Doe', 'M', 180, 75);
INSERT INTO PARTICIPANT VALUES (2, 2, 102, 'Jane', 'Smith', 'F', 165, 55);
INSERT INTO PARTICIPANT VALUES (3, 3, 103, 'Michael', 'Johnson', 'M', 175, 70);
INSERT INTO PARTICIPANT VALUES (4, 4, 104, 'Emily', 'Davis', 'F', 160, 50);
INSERT INTO PARTICIPANT VALUES (5, 5, 105, 'Chris', 'Brown', 'M', 190, 80);
INSERT INTO PARTICIPANT VALUES (6, 1, 101, 'Sarah', 'Wilson', 'F', 170, 60);
INSERT INTO PARTICIPANT VALUES (7, 2, 102, 'David', 'Lee', 'M', 175, 70);
```

```
CREATE TABLE NATION (
    nation_id NUMBER,
    country_name VARCHAR2(30),
    population NUMBER
);

INSERT INTO NATION VALUES (1, 'United States', 331002651);
INSERT INTO NATION VALUES (2, 'India', 1380004385);
INSERT INTO NATION VALUES (3, 'China', 1444216107);
INSERT INTO NATION VALUES (4, 'Brazil', 212559417);
INSERT INTO NATION VALUES (5, 'United Kingdom', 67886011);
```

> 아래와 같이 이름과 성별, 그리고 국가, 인구를
> PARTICIPANT 테이블과 NATION 테이블을 이용하여 조회하는 SQL문을 작성해 보세요.

FIRST_NAME	GENDER	COUNTRY_NAME	POPULATION
John	M	United States	331002651
Jane	F	India	1380004385
Michael	M	China	1444216107
Emily	F	Brazil	212559417
Chris	M	United Kingdom	67886011
Sarah	F	United States	331002651
David	M	India	1380004385

```
SELECT p.first_name,
       p.gender,
       n.country_name,
       n.population
FROM participant p, nation n
WHERE p.nation_id = n.nation_id;
```

(2) 사원-부서 WHERE 절 검색 조건 사례

- WHERE 절에서는 JOIN 조건과 더불어 추가적인 제한 조건도 덧붙여서 사용할 수 있습니다.
- 예시

```
SELECT e.ENAME,
       e.JOB,
       d.DNAME
FROM emp e, dept d
WHERE e.DEPTNO = d.DEPTNO
  AND e.SAL = 3000
ORDER BY e.ENAME;
```

- 실습

> 여성 참가자들의 이름과 국적 및 인구 수를 이름 내림차순으로 조회하는 SQL문을 작성해 보세요.

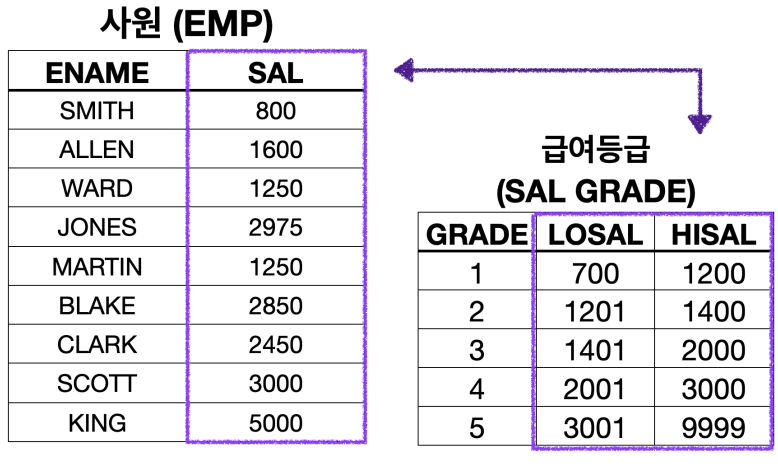
FIRST_NAME	COUNTRY_NAME	POPULATION
Sarah	United States	331002651
Jane	India	1380004385
Emily	Brazil	212559417

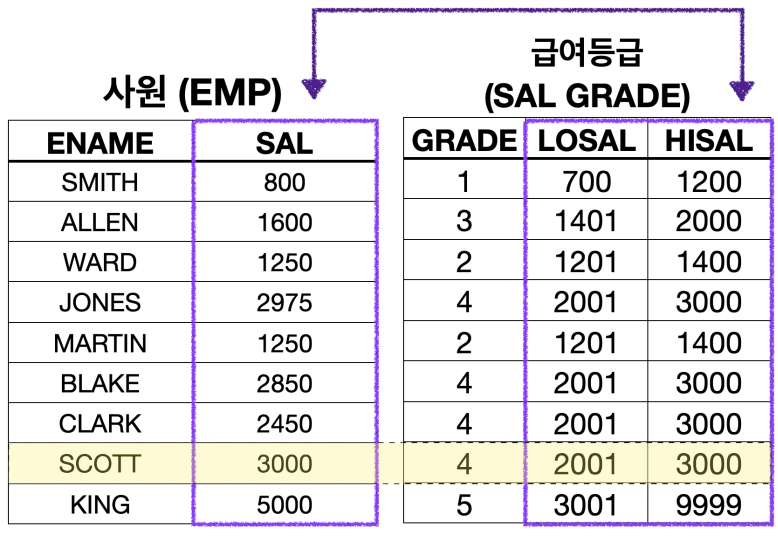
```
SELECT p.first_name,
       n.country_name,
       n.population
FROM participant p, nation n
WHERE p.nation_id = n.nation_id AND p.gender = 'F'
ORDER BY p.first_name DESC;
```

▼ 2) Non EQUI JOIN

Non EQUI(비등가) JOIN은 두 개의 테이블 간에 칼럼들이 서로 정확하게 일치하는 것이 아닌 특정 범위 내에 있는 경우 JOIN을 하고자 할 때 사용합니다. 그래서 `=` 연산자가 아닌 다른 (`BETWEEN` , `>` , `>=` , `<` , `<=`) 연산자들을 사용을 합니다. 다만 설계상의 이유로 Non EQUI JOIN을 수행하지 못하는 경우도 있습니다.

- 예시

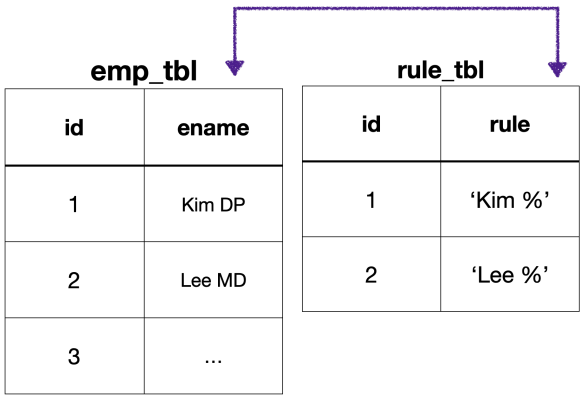





```
SELECT e.ename '사원명', e.sal '급여', s.grade '급여등급'
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

또한 LIKE 연산으로 다른 테이블의 값을 이용하여 JOIN을 나타낼 수 있습니다.

- 예시




```
SELECT COUNT(*) AS CNT
FROM emp_tbl a, rule_tbl b
WHERE a.ename LIKE b.rule;
```

- 실습

- 테이블 구조

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY
1	Alice	50000
2	Bob	60000
3	Charlie	70000

EMPLOYEE_ID	BONUS_AMOUNT
1	2000
3	1500

- 아래의 SQL문을 실행하면 어떤 결과가 나올까?

```
SELECT e.employee_id, e.employee_name, e.salary, b.bonus_amount
FROM employees e, bonuses b
WHERE e.salary > 55000;
```

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	BONUS_AMOUNT
2	Bob	60000	2000
2	Bob	60000	1500
3	Charlie	70000	2000
3	Charlie	70000	1500

결과 테이블

두개의 테이블을 JOIN 하기 위해서는 적어도 하나의 공통된 컬럼이 필요하며, JOIN 조건이 정의되지 않았거나 잘못된 경우, 또는 하나의 테이블이 다른 테이블의 모든 행과 JOIN이 된다면 **Cartesian Product (카티션 곱)** 즉 모든 행의 조합이 나오게 된다.

(컴퓨터는 시키는대로 잘한다)

원하는 결과를 얻기 위해서는

```
SELECT e.employee_id, e.employee_name, e.salary, b.bonus_amount
FROM employees e, bonuses b
WHERE e.employee_id = b.employee_id AND e.salary > 55000;
```

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	BONUS_AMOUNT
3	Charlie	70000	1500

▼ 3) 3개 이상 TABLE JOIN

3개 이상의 테이블을 JOIN을 할 때는 FROM 절에 JOIN 하고자 하는 테이블을 차례대로 작성해 주면 됩니다. 그러면 첫 번째와 두 번째 순서의 테이블이 먼저 JOIN 이 완료되고 그다음 순서로 세 번째 테이블과의 JOIN이 이루어지면서 3개 테이블의 JOIN이 완료됩니다. (JOIN은 항상 두개의 테이블에서 일어난다)

- 예시

```
SELECT P.PLAYER_NAME  '선수명',
       P.POSITION     '포지션',
       T.REGION_NAME  '연고지',
       T.TEAM_NAME    '팀명',
       S.STADIUM_NAME '구장명'
FROM PLAYER P,
     TEAM T,
     STADIUM S
```

```
WHERE P.TEAM_ID = T.TEAM_ID
  AND T.STADIUM_ID = S.STADIUM_ID
ORDER BY '선수명';
```

- 모든 테이블을 정규화 시켜서 JOIN으로 데이터를 찾기에는 JOIN 진행시 시스템의 부하가 큰 부분이 단점입니다. 그래서 무작정 정규화하기보다는 역정규화 하여 최적의 테이블 구조를 만드는 것이 중요합니다.

- 실습

> 참가자들의 이름, 국적, 주 종목을 조회하고 '주 종목' 내림차순으로 정렬해보세요.

```
SELECT p.first_name, n.country_name, s.sport_name
FROM participant p, nation n, sport s
WHERE p.nation_id = n.nation_id AND p.main_sport_id = s.sport_id
ORDER BY s.sport_name;
```

▼ 연습문제

☑️ 문제 1

Q. 문제	**3개의 테이블을 조인하여 결과를 출력하고자 할 때, 3개의 테이블을 조인하기 위해 필요한 조인 조건의 개수는 최소 몇 개인가?**
A. (1)	1
A. (2)	2
A. (3)	3
A. (4)	4

▼ 정답

(2) 최소로 필요한 조인 조건의 개수는 '테이블의 개수 - 1'이다

☑️ 문제 2

Q. 문제	**조인에 대한 설명으로 적절한 것은?**
A. (1)	조인은 다른 데이터베이스 시스템 간의 데이터 이동을 위해 사용된다
A. (2)	관련된 데이터를 단 하나의 테이블에서 결합하여 검색하는 것을 조인이라고 한다
A. (3)	데이터 그룹화하고 정렬하는 데 조인이 필수로 사용된다
A. (4)	2개 이상의 집합을 결합하여 데이터를 출력하는 것을 조인이라고 한다

▼ 정답

(4)

☑️ 문제 3

Q. 문제	**EQUI 조인에 대한 설명으로 옳지 않은 것은?**
A. (1)	INNER JOIN 을 사용하여 JOIN을 만들 수 있다
A. (2)	두 개의 테이블 간에 교집합을 구한다
A. (3)	`=` 비교 연산자를 사용한다
A. (4)	모든 비교 연산자를 사용해서 조인할 수 있다.

▼ 정답

(4) EQUI 조인은 `=`을 사용해서 조인한다

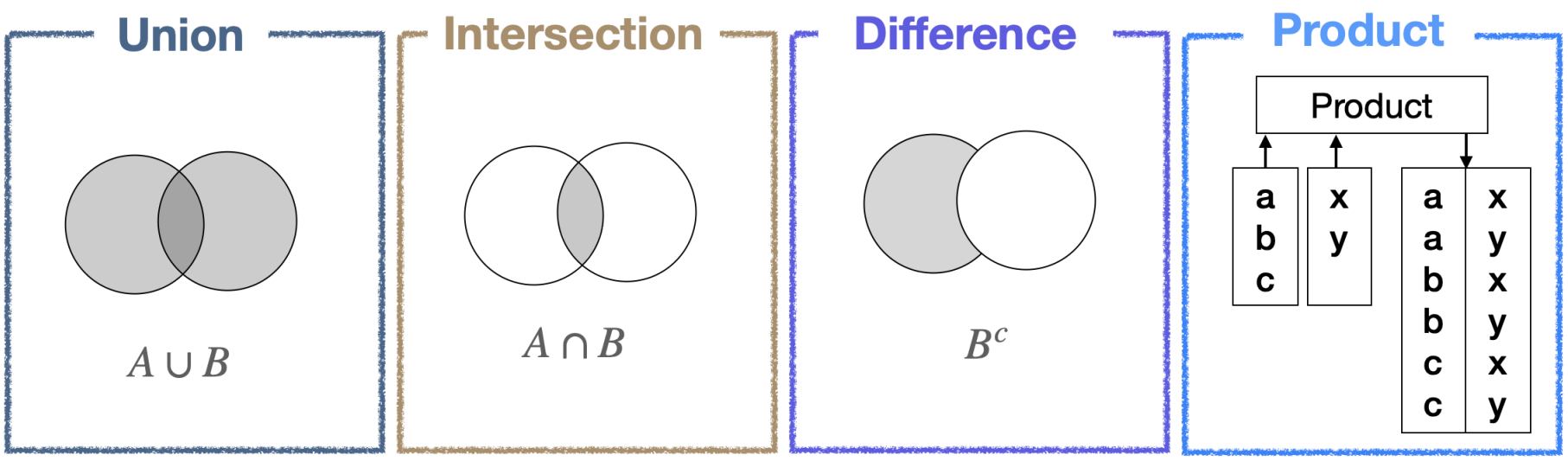
02. 표준 조인

✔️ **ANSI/ISO SQL에서 규정한 JOIN 문법에 대해 학습합니다.**

▼ 1) STANDARD SQL 개요

현재 사용하는 집합 연산을 토대로 만든 관계형 대수들은 4개의 일반 집합 연산자와 4개의 순수 관계 연산자로 나눌 수 있습니다. SQL은 8개의 연산자를 활용한 연산을 구현하는 데 초점을 두고 진행합니다.

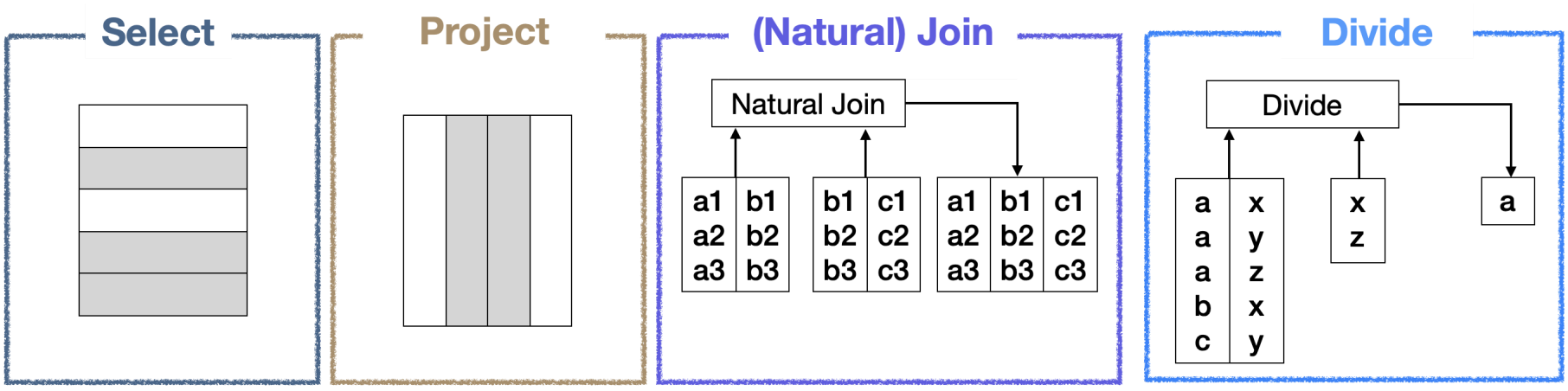
☑️ 일반 집합 연산자



- **UNION 연산**
  - 수학의 합집합으로 제공하기 위한 연산자입니다.
  - 공통 교집합의 중복을 없애기 위한 사전 작업으로 시스템의 부하를 주는 정렬 작업이 발생합니다.
  - UNION ALL
    - 공통 집합을 중복해서 그대로 보여주는 연산자입니다.
    - 데이터 정렬 및 중복 작업을 하지 않기 때문에 같은 결과라면 UNION보다 효율이 좋습니다.
    - 그대로 보여주기 때문에 정렬 작업이 일어나지 않으며 응답 속도 향상이나 자원 효율화 측면에서 UNION ALL 사용이 권장됩니다.
- **INTERSECTION 연산**
  - 수학의 교집합으로 두 집합의 공통된 값을 추출할 때 사용하는 연산자입니다.
  - 두 집합의 공통된 요소(공통 행)를 출력합니다.
- **DIFFERENCE 연산**
  - 수학의 차집합으로 첫 번째 집합에서 두 번째 집합과의 공통집합을 제외한 부분을 출력하기 위해 사용되는 연산자 입니다.
  - Oracle에서는 MINUS로 사용하며 ANSI 표준 대부분의 DBMS 제공 회사는 EXCEPT로 사용합니다.
- **PRODUCT 연산**
  - 수학에서의 곱집합으로 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말합니다.
  - 만약 양쪽 테이블에 자료가 각각 M개 N개가 있으면 M*N 건의 데이터 조합이 발생합니다.
  - CARTESIAN PRODUCT(데카르트의 곱)이라고도 합니다.

☑️ 순수 관계 연산자

순수 관계 연산자는 관계형 데이터베이스를 구현하기 위해 새롭게 만들어진 연산자입니다.



- **SELECT 연산**
  - SELECT 절과 키워드는 같으나 동작은 다릅니다.
  - SELECT 문에서는 WHERE 절로 표현 가능합니다.
  - 특정 행에 대한 부분 집합이라고 할 수 있습니다.
  - SELECT 연산자는 특정 조건을 만족하는 데이터들을 뽑아서 새로운 테이블을 만들어서 사용하고 싶을 때 이용하는 연산자입니다.
- **PROJECT 연산**
  - 특정 열에 대한 부분 집합이라고 할 수 있습니다.
  - SELECT는 특정 조건을 만족하는 데이터라면 PROJECT는 원하는 칼럼을 뽑아서 새로운 테이블을 만들어서 사용하고 싶을 때 사용하는 연산자입니다.
- **JOIN 연산**
  - JOIN은 공통 속성을 중심으로 두 개의 테이블을 하나로 합쳐서 새로운 테이블을 만드는 연산자입니다.
  - INNER JOIN 조건과 함께 FROM 절의 NATURAL JOIN, OUTER JOIN, USING 조건절, ON 조건절 등으로 가장 다양하게 발전한 연산자입니다.
- **DIVIDE 연산**
  - 테이블 A와 B가 있다고 가정해 보겠습니다. 여기에서 테이블 B는 테이블 A에 속해 있습니다. 이때 테이블 A와 B의 DIVIDE 연산의 결과는 다음과 같습니다.
    - A에 속해있는 B의 값을 전부 가지고 있는 데이터를 구합니다.
    - 그리고 여기에서 B의 칼럼을 제외한 값이 DIVIDE 연산의 결과입니다.
  - 현재는 SQL문으로 구현되어 있지 않은 연산자입니다.

▼ 2) 대표적인 JOIN의 6가지 방식

2개 이상의 테이블을 합쳐서 원하는 결과를 만들어 출력하는 것을 '조인(JOIN)'이라고 합니다. JOIN은 관계형 데이터베이스에서 가장 핵심이기 때문에 ANSI/ISO SQL에서 규정한 JOIN 문법은 WHERE 절을 이용한 JOIN과 차이가 있습니다. 사용자는 기존 WHERE 절의 검색 조건과 테이블 간의 JOIN 조건을 구분 없이 사용하던 방식을 그대로 사용할 수 있습니다. 동시에 추가된 선택 기능으로 JOIN 조건을 FROM 절에서 명시적으로도 정의할 수 있습니다. FROM 절에 명시적으로 사용할 수 있는 JOIN의 종류는 대표적으로 하나씩 알아가 보겠습니다.

✅ 대표적인 JOIN 방법 6가지

조인 형태	설명
INNER JOIN	- JOIN 조건에서 일치하는 행만 반환 - EQUI 조인이라고도 함
NATURAL JOIN	- 두 테이블 간의 동일한 이름을 갖는 모든 칼럼에 대해 INNER JOIN 수행 - Oracle에서만 지원하며 SQL Server에서는 지원하지 않음
USING 조건절	- NATURAL JOIN에 FROM 절에서 USING 조건을 이용하면 같은 이름을 가진 칼럼에서 원하는 칼럼만 선택적으로 INNER JOIN 가능 - Oracle에서만 지원하며 SQL Server에서는 지원하지 않음
ON 조건절	- 명시적으로 JOIN 조건을 지정하는 데 사용 - 칼럼명이 달라도 JOIN 조건을 사용할 수 있다

조인 형태	설명
CROSS JOIN	- PRODUCT의 개념으로 테이블 간의 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 의미
OUTER JOIN	- INNER JOIN과 대비하여 JOIN 조건에서 동일한 값이 없는 행도 결과 집합에 포함시킬 때 사용

(1) INNER JOIN

- JOIN 조건에서 일치하는(동일한 값이 있는) 행만 반환합니다.
- INNER JOIN은 그동안 WHERE 절에서 사용하던 JOIN 조건을 FROM 절에서 정의하겠다는 표시입니다.
- 반드시 USING 조건절이나 ON 조건절을 사용해줘야 합니다.

- 기본 구조
  - INNER 키워드는 생략 가능합니다.

```
SELECT 테이블1.칼럼1, 테이블2.칼럼2, ..
FROM 테이블1 [INNER] JOIN 테이블2
  ON 테이블1.칼럼 = 테이블2.칼럼;
```

- 예시

```
SELECT
  emp.deptno,
  empno,
  ename,
  dname
FROM emp JOIN dept
  ON emp.deptno = dept.deptno;
```

- 실습

▼ 실습 데이터

```
CREATE TABLE VolunteerActivity (
    activity_id NUMBER PRIMARY KEY,
    juice_type VARCHAR2(20),
    quantity_sold NUMBER
);

INSERT INTO VolunteerActivity VALUES (1, 'Orange Juice', 100);
INSERT INTO VolunteerActivity VALUES (2, 'Apple Juice', 80);
INSERT INTO VolunteerActivity VALUES (3, 'Grape Juice', 640);
INSERT INTO VolunteerActivity VALUES (4, 'Pineapple Juice', 300);
INSERT INTO VolunteerActivity VALUES (5, 'Lemonade Juice', 110);
```

```
CREATE TABLE Unit (
    unit_id NUMBER PRIMARY KEY,
    unit_name VARCHAR2(20),
    activity_id NUMBER,
    FOREIGN KEY (activity_id) REFERENCES VolunteerActivity(activity_id)
);

INSERT INTO Unit VALUES (101, 'Alpha Unit', 1);
INSERT INTO Unit VALUES (102, 'Bravo Unit', 2);
INSERT INTO Unit VALUES (103, 'Charlie Unit', 3);
INSERT INTO Unit VALUES (104, 'Delta Unit', 4);
INSERT INTO Unit VALUES (105, 'Echo Unit', 5);
```

```
CREATE TABLE Soldier (
    soldier_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(20),
    last_name VARCHAR2(20),
```

```
    rank VARCHAR2(20),
    unit_id NUMBER,
    activity_id NUMBER,
    FOREIGN KEY (unit_id) REFERENCES Unit(unit_id),
    FOREIGN KEY (activity_id) REFERENCES VolunteerActivity(activity_id)
);


INSERT INTO Soldier VALUES (3028, 'John', 'Doe', 'Sergeant', 101, 1);
INSERT INTO Soldier VALUES (2734, 'Jane', 'Smith', 'Private', 102, 2);
INSERT INTO Soldier VALUES (3103, 'Michael', 'Johnson', 'Private', 103, 3);
INSERT INTO Soldier VALUES (4865, 'Emily', 'Davis', 'Private', 104, 4);
INSERT INTO Soldier VALUES (5371, 'Chris', 'Brown', 'Sergeant', 105, 5);
```

> 부대 이름과 봉사활동 시 판매했던 과일명을 출력해 보세요.

```
SELECT u.unit_name, v.juice_type
FROM unit u JOIN volunteerActivity v
  ON u.activity_id = v.activity_id;
```

(2) NATURAL JOIN

- NATURAL JOIN은 INNER JOIN의 하위 개념으로 두 테이블의 JOIN 되는 칼럼을 별도로 지정하지 않아도 동일한 이름을 갖는 모든 칼럼들에 대해 자동으로 EQUI JOIN을 수행합니다.
- NATURAL JOIN 사용시 주의해야할 점은 JOIN 기준이되는 칼럼에 ALIAS나 테이블 명과 같은 접두사를 칼럼에 붙일 수 없습니다. 그리고 동일한 칼럼명이라도 다른 데이터 유형이 저장되었다면 제대로 동작되지 않게 됩니다. 반드시 데이터 유형이 같은지 확인을 해야합니다.
- 해당 JOIN은 SQL Server 에서는 지원하지 않는 기능입니다.

- 기본 구조

```
SELECT 칼럼1, 칼럼2, ..
FROM 테이블1 NATURAL JOIN 테이블2;
```

- 예시

```
SELECT depno, empno, ename, dname
FROM emp NATURAL JOIN dept;
```

- 실습

> 군인 이름, 부대 이름, 계급을 Soldier, Unit 테이블의 NATURAL JOIN을 이용하여 출력해보세요.

```
SELECT first_name, unit_name, rank
FROM Soldier NATURAL JOIN Unit;
```

(3) USING 조건절

- FROM 절의 USING 조건절을 이용하면 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI JOIN을 할 수 있습니다.
- 결과 집합이 INNER JOIN을 수행한 결과와 동일합니다.
- NATURAL JOIN 절에서 USING 절을 이용하려면 NATURAL JOIN을 명시하지 않은 상태로 USING절을 사용해야 합니다.

- USING 조건절도 NATURAL JOIN과 마찬가지로 JOIN 칼럼에 대해서 ALIAS 나 테이블 이름과 같은 접두사를 붙일 수 없습니다. 다만, JOIN 칼럼이 아닌 다른 칼럼은 ALIAS 사용이 가능합니다.
- SQL Server에서는 지원하지 않습니다.

- 기본 구조

```
SELECT 칼럼1, 칼럼2, ..
FROM 테이블1 JOIN 테이블2
  USING (EXPR);
```

- 예시

```
SELECT empno,  -- JOIN 컬럼에 대해 Alias 사용 불가
       emp.ename,
       dept.loc,
       dept.dname
FROM emp JOIN dept
  USING (deptno); -- 괄호 없으면 에러 발생
```

- 실습

> Soldier와 Unit 테이블을 USING절을 이용하여 부대 ID, 군인이름, 부대이름, 계급을 출력해보세요.

```
SELECT unit_id,
       s.first_name,
       u.unit_name,
       s.rank
FROM Soldier s JOIN Unit u
USING (unit_id);
```

(4) ON 조건절

- USING 조건절은 같은 칼럼명일 때 JOIN 하려는 칼럼을 선택하는 것이라면 ON 조건절은 칼럼명이 다르더라도 JOIN 조건을 사용할 수 있습니다.
- NATURAL JOIN에서 임의의 JOIN 조건을 지정하거나 이름이 다른 칼럼명을 JOIN 조건으로 사용하고 싶을 때 혹은 JOIN 칼럼을 명시하기 위해서는 ON 조건절을 사용합니다.
- ON 조건절을 사용하는 JOIN의 경우는 ALIAS나 테이블명을 접두사로 붙여도 상관없으며 오히려 논리적으로 명확하게 지정해 주는 것이 좋습니다. 만약 동일한 칼럼명을 기재할 경우 반드시 ALIAS를 통한 구분이 필요합니다.

✅ ON 조건절에 활용법

1. **WHERE 절과의 혼용**

   ON 조건절은 WHERE 절과 충돌 없이 사용할 수 있습니다.

   - 예시

   ```
   SELECT e.ename, e.deptno, d.deptno, d.dname
   FROM emp e JOIN dept d
     ON e.deptno = d.deptno
   WHERE e.deptno = 30;
   ```

   - 실습 1

   > 부대 이름이 Alpha Unit 인 군인의 이름과 성(last_name), 계급 그리고 봉사활동 ID를 출력 해보세요.

```
SELECT s.first_name, s.last_name, s.rank, u.activity_id
FROM Soldier s JOIN Unit u
  ON s.activity_id = u.activity_id
WHERE u.unit_name = 'Alpha Unit';
```

2. **ON 조건절 + 데이터 검증 조건 추가**

   ON 조건절에서 JOIN 조건 외에도 데이터를 검색하기 위한 조건을 추가할 수 있습니다. 검색 조건 목적인 경우에는 WHERE 절을 사용하는 것을 권장합니다.

   - 예시

   ```
   SELECT e.ename, e.deptno, d.deptno, d.dname
   FROM emp e JOIN dept d
     ON e.deptno = d.deptno AND e.mgr = 7698; -- WHERE 사용 권장
   ```

   - 실습 2

   > 주스 판매량이 110 이상인 부대의 이름과 판매량을 출력해 보세요.

   ```
   SELECT u.unit_name, v.quantity_sold
   FROM VolunteerActivity v JOIN Unit u
     ON v.activity_id = u.activity_id AND v.quantity_sold >= 110;
   ```

3. **ON 조건절 예제**

   같은 의미이지만 다른 이름의 칼럼을 사용할 때는 USING 조건절을 사용할 수 없으며 이때는 ON 조건절을 사용해야 합니다.

   - 예시
     - team_id 와 hometeam_id는 동일한 의미를 가진 값을 가지고 있습니다.

   ```
   SELECT team_name, team_id, stadium_name
   FROM team JOIN stadium
     ON team.team_id = stadium.hometeam_id
   ORDER BY team_id;
   ```

   - 실습 3

   > 군인의 이름과 부대명을 출력해보세요.

   ```
   SELECT first_name, unit_name
   FROM Soldier s JOIN Unit u
     ON s.unit_id = u.unit_id
   ORDER BY unit_name;
   ```

4. **다중 테이블 JOIN**

   세 개 이상의 테이블을 JOIN을 하려고 할 때는 ON 조건절 다음에 JOIN 을 중첩적으로 명시해 주면 됩니다.

   - 예시

   ▼ dept_temp 데이터

   ```
   CREATE TABLE dept_temp (
   deptno NUMBER PRIMARY KEY,
   dname VARCHAR2(30),
   loc VARCHAR2(30)
   );

   INSERT INTO dept_temp VALUES (10, 'Administration', 'New York');
   ```

```
INSERT INTO dept_temp VALUES (20, 'Marketing', 'Los Angeles');
INSERT INTO dept_temp VALUES (30, 'Purchasing', 'San Francisco');
INSERT INTO dept_temp VALUES (40, 'Human Resources', 'Chicago');
```

- 사원과 dept 테이블의 소속 부서명, dept_temp 테이블의 바뀐 부서명 정보를 출력

```
SELECT e.empno, d.deptno, d.dname, t.dname AS new_dname
FROM emp e JOIN dept d
  ON e.deptno = d.deptno JOIN dept_temp t
  ON e.deptno = t.deptno;
```

- 실습

> 주스 판매량이 100에서 600사이인 부대의 이름과 판매량, 주스이름, 해당 부대에 소속된 군인 이름, 계급을 출력해 보세요.

```
SELECT u.unit_name, v.juice_type, v.quantity_sold, s.first_name, s.rank
FROM VolunteerActivity v JOIN Unit u
  ON v.activity_id = u.activity_id JOIN Soldier s
  ON u.unit_id = s.unit_id
WHERE v.quantity_sold BETWEEN 100 AND 600;
```

✅ 3개 이상의 테이블에 대한 JOIN

만약 3개 이상의 테이블을 JOIN 하려고 한다면 조건은 최소 2개가 필요합니다.

최소 JOIN 조건의 수 = JOIN 테이블의 수 - 1

(5) CROSS JOIN

- 일반 집합 연산자의 PRODUCT 개념으로 테이블 간 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 결과로 나타냅니다. 두 테이블에 대한 CARTESIAN PRODUCT 또는 CROSS PRODUCT 와 같은 표현으로도 불리며 결과는 M*N 건의 데이터 조합이 발생하게 됩니다.
- CROSS JOIN 은 WHERE 절에 JOIN 조건을 추가하여 사용할 수 있습니다. 다만 INNER JOIN과 같은 결과를 얻기 때문에 CROSS JOIN 을 사용하는 의미가 없어지게 됩니다.
- 평소에는 CROSS JOIN이 필요한 경우는 많지 않지만, 간혹 튜닝이나 리포트를 작성하기 위해 의도적으로 사용하는 경우가 있습니다.

- 예시

```
SELECT ename, dname
FROM emp CROSS JOIN dept
ORDER BY ename;
```

- 실습

> 부대 테이블과 봉사활동 테이블을 CROSS JOIN을 활용하여 부대 이름, 주스 종류, 판매수량을 판매수량 내림차순으로 하여 출력해 보세요.

```
SELECT unit_name, juice_type, quantity_sold
FROM Unit CROSS JOIN VolunteerActivity
ORDER BY quantity_sold DESC;
```

(6) OUTER JOIN

- OUTER JOIN은 "OUTER 집합을 기준으로 JOIN"한다는 의미입니다. OUTER 테이블은 모두 출력되고 INNER 테이블은 매칭되는 데이터만 출력한다는 의미를 갖습니다.
- OUTER JOIN은 JOIN 조건에서 동일한 값이 없는 데이터도 반환할 때 사용할 수 있으며 INNER JOIN과 대비되는 JOIN 입니다. OUTER JOIN도 INNER JOIN과 마찬가지로 JOIN 조건을 FROM 절에서 정의하는 것이기 때문에 USING 조건절이나 ON 조건절을 반드시 사용해야 합니다.
- OUTER JOIN은 기준이 되는 테이블에 따라 LEFT OUTER JOIN / RIGHT OUTER JOIN 으로 나뉘며 기준이 되는 테이블이 조인 수행시 무조건 드라이빙 테이블이 됩니다.
- 과거 OUTER JOIN을 위해 Oracle은 JOIN 칼럼 뒤에 (+)를 붙여서 표시하였습니다.

> 💡 **드라이빙 테이블(Driving Table)이란?**
> 드라이빙 테이블은 데이터 행이 먼저 선택되는 테이블입니다.

1. **LEFT OUTER JOIN**

테이블 A와 B가 있을 때 LEFT OUTER JOIN을 수행할 때 테이블 A가 기준이 되어 먼저 데이터를 읽어옵니다. 그 후에 B의 데이터를 읽어온 다음 A와의 JOIN 칼럼과 비교하여 같은 값이 있다면 해당 데이터를 가져오고, 값이 없다면 NULL 값으로 없는 데이터를 채워 넣은 테이블을 생성합니다. INNER JOIN 처럼 OUTER JOIN 도 OUTER 키워드를 생략할 수 있습니다.

- 예시

▼ 예시 테이블 데이터

```
CREATE TABLE DEPT_EX (
  DEPT_NO CHAR(4)
, DEPT_NM VARCHAR2(50) NOT NULL
, CONSTRAINT DEPT_EX_PK PRIMARY KEY (DEPT_NO) );

INSERT INTO DEPT_EX VALUES ('1001', '마케팅팀');
INSERT INTO DEPT_EX VALUES ('1002', '개발팀');
INSERT INTO DEPT_EX VALUES ('1003', '디자인팀' ); COMMIT;


CREATE TABLE EMP_EX (
  EMP_NO CHAR(5)
, EMP_NM VARCHAR2(50) NOT NULL
, DEPT_NO CHAR(4)
, CONSTRAINT EMP_EX_PK PRIMARY KEY (EMP_NO) );
INSERT INTO EMP_EX VALUES ('00001', 'Jane', '1001');
INSERT INTO EMP_EX VALUES ('00002', 'Lisa', '1002');
INSERT INTO EMP_EX VALUES ('00003', 'Jake', '1002');
INSERT INTO EMP_EX VALUES ('00004', 'Mark', '1003');
COMMIT;
```

```
SELECT A.DEPT_NO , A.DEPT_NM
  , NVL(B.EMP_NO, 'NULL') AS EMP_NO
  , NVL(B.EMP_NM, 'NULL') AS EMP_NM
FROM DEPT_EX A LEFT OUTER JOIN EMP_EX B
  ON (A.DEPT_NO = B.DEPT_NO AND A.DEPT_NM = '개발팀')
WHERE A.DEPT_NO IS NOT NULL
```

- 실습

▼ 실습 데이터

```
CREATE TABLE VolunteerActivity (
    activity_id NUMBER PRIMARY KEY,
    juice_type VARCHAR2(20),
    quantity_sold NUMBER
);

INSERT INTO VolunteerActivity VALUES (1, 'Orange Juice', 100);
INSERT INTO VolunteerActivity VALUES (2, 'Apple Juice', 80);
```

```
INSERT INTO VolunteerActivity VALUES (3, 'Grape Juice', 640);
INSERT INTO VolunteerActivity VALUES (4, 'Pineapple Juice', 300);
INSERT INTO VolunteerActivity VALUES (5, 'Lemonade', 110);
```

```
CREATE TABLE Unit (
    unit_id NUMBER PRIMARY KEY,
    unit_name VARCHAR2(20),
    activity_id NUMBER,
    FOREIGN KEY (activity_id) REFERENCES VolunteerActivity(activity_id)
);

INSERT INTO Unit VALUES (101, 'Alpha Unit', 1);
INSERT INTO Unit VALUES (102, 'Bravo Unit', 2);
INSERT INTO Unit VALUES (103, 'Charlie Unit', 3);
INSERT INTO Unit VALUES (104, 'Delta Unit', 4);
INSERT INTO Unit VALUES (105, 'Echo Unit', 5);
INSERT INTO Unit VALUES (106, 'Grenadier Unit', 5);
INSERT INTO Unit VALUES (107, 'Parachute Unit', 4);
```

```
CREATE TABLE Soldier (
    soldier_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(20),
    last_name VARCHAR2(20),
    rank VARCHAR2(20),
    unit_id NUMBER,
    activity_id NUMBER,
    FOREIGN KEY (unit_id) REFERENCES Unit(unit_id),
    FOREIGN KEY (activity_id) REFERENCES VolunteerActivity(activity_id)
);

INSERT INTO Soldier VALUES (3028, 'John', 'Doe', 'Sergeant', 101, 1);
INSERT INTO Soldier VALUES (2734, 'Jane', 'Smith', 'Private', 102, 2);
INSERT INTO Soldier VALUES (3103, 'Michael', 'Johnson', 'Private', 103, 3);
INSERT INTO Soldier VALUES (4865, 'Emily', 'Davis', 'Private', 104, 4);
INSERT INTO Soldier VALUES (5371, 'Chris', 'Brown', 'Sergeant', 105, 5);
INSERT INTO Soldier VALUES (0297, 'Eleanor', 'Grace', 'Sergeant',NULL, NULL);
INSERT INTO Soldier VALUES (9234, 'Alexander', 'James', 'Sergeant',NULL, NULL);
```

> 부대를 기준으로 해당 부대에 속하는 군인의 정보를 나타내보세요. 해당 부대에 속한 군인이 없다면 빈 값으로 나타내세요.

```
-- 바꿔가며 비교해보기
SELECT U.unit_id
     , U.unit_name
     , S.first_name
FROM Unit U LEFT OUTER JOIN Soldier S
ON (U.unit_id = S.unit_id)
ORDER BY unit_id
;
```

2. **RIGHT OUTER JOIN**

   LEFT OUTER JOIN 과는 반대로 우측에 있는 테이블이 기준이 되어 결과를 생성하는 JOIN 방법입니다.

   - 예시
     - 사원이 없는 부서 정보도 같이 출력하는 예시

     ```
     SELECT e.ename, d.deptno, d.dname
     FROM emp e RIGHT [OUTER] JOIN dept d
       ON e.deptno = d.deptno;
     ```

   - 실습

     ▼ 실습 데이터

```
CREATE TABLE VolunteerActivity (
    activity_id NUMBER PRIMARY KEY,
    juice_type VARCHAR2(20),
    quantity_sold NUMBER
);

INSERT INTO VolunteerActivity VALUES (1, 'Orange Juice', 100);
INSERT INTO VolunteerActivity VALUES (2, 'Apple Juice', 80);
INSERT INTO VolunteerActivity VALUES (3, 'Grape Juice', 640);
INSERT INTO VolunteerActivity VALUES (4, 'Pineapple Juice', 300);
INSERT INTO VolunteerActivity VALUES (5, 'Lemonade', 110);
```

```
CREATE TABLE Unit (
    unit_id NUMBER PRIMARY KEY,
    unit_name VARCHAR2(20),
    activity_id NUMBER,
    FOREIGN KEY (activity_id) REFERENCES VolunteerActivity(activity_id)
);

INSERT INTO Unit VALUES (101, 'Alpha Unit', 1);
INSERT INTO Unit VALUES (102, 'Bravo Unit', 2);
INSERT INTO Unit VALUES (103, 'Charlie Unit', 3);
INSERT INTO Unit VALUES (104, 'Delta Unit', 4);
INSERT INTO Unit VALUES (105, 'Echo Unit', 5);
INSERT INTO Unit VALUES (106, 'Grenadier Unit', 5);
INSERT INTO Unit VALUES (107, 'Parachute Unit', 4);
```

```
CREATE TABLE Soldier (
    soldier_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(20),
    last_name VARCHAR2(20),
    rank VARCHAR2(20),
    unit_id NUMBER,
    activity_id NUMBER,
    FOREIGN KEY (unit_id) REFERENCES Unit(unit_id),
    FOREIGN KEY (activity_id) REFERENCES VolunteerActivity(activity_id)
);

INSERT INTO Soldier VALUES (3028, 'John', 'Doe', 'Sergeant', 101, 1);
INSERT INTO Soldier VALUES (2734, 'Jane', 'Smith', 'Private', 102, 2);
INSERT INTO Soldier VALUES (3103, 'Michael', 'Johnson', 'Private', 103, 3);
INSERT INTO Soldier VALUES (4865, 'Emily', 'Davis', 'Private', 104, 4);
INSERT INTO Soldier VALUES (5371, 'Chris', 'Brown', 'Sergeant', 105, 5);
INSERT INTO Soldier VALUES (0297, 'Eleanor', 'Grace', 'Sergeant',NULL, NULL);
INSERT INTO Soldier VALUES (9234, 'Alexander', 'James', 'Sergeant',NULL, NULL);
```

> 각 군인들은 어떤 부대에 속해있는지 RIGHT JOIN을 활용하여 나타내보세요.

```
SELECT S.first_name, U.unit_name
FROM Unit U RIGHT JOIN Soldier S
  ON S.unit_id = U.unit_id
ORDER BY U.unit_name;
```

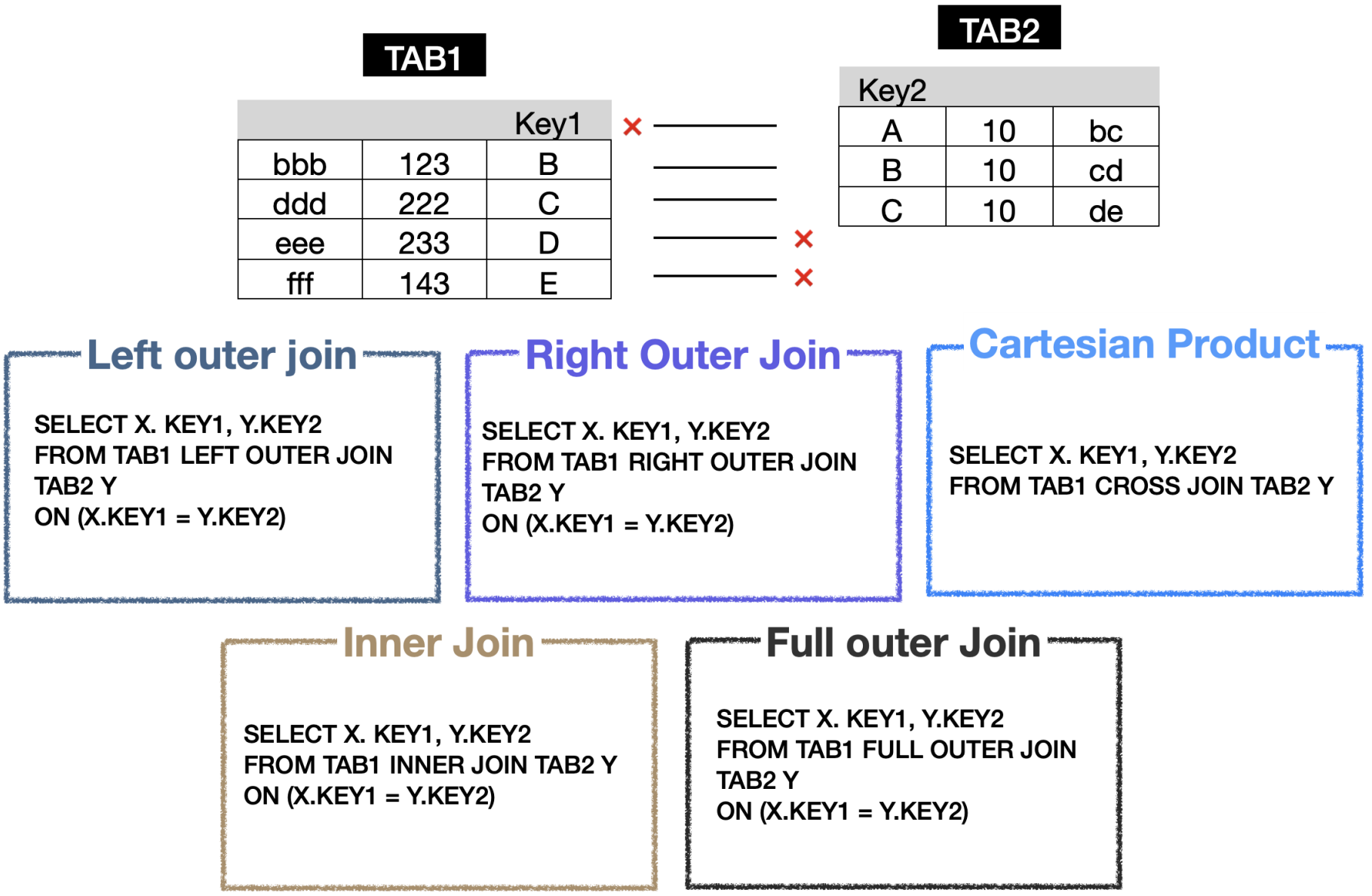
3. **FULL OUTER JOIN**

   FULL OUTER JOIN은 좌측 테이블과 우측 테이블이 동시에 기준이 되는 JOIN 방법으로 LEFT OUTER JOIN과 RIGHT OUTER JOIN의 결과를 합집합으로 처리한 결과와 동일합니다.

   - 예시
     - deptno 칼럼을 기준으로 dept와 dept_temp 테이블을 FULL JOIN 하는 방법

```
SELECT *
FROM dept FULL [OUTER] JOIN dept_temp
  ON dept.deptno = dept_temp.deptno;
```

▼ 3) INNER vs OUTER vs CROSS JOIN 비교



- **JOIN 종류 정리**

JOIN 종류	설명
INNER JOIN	양쪽 테이블에 모두 존재하는 키 값이 B-B, C-C인 2건이 출력됩니다.
LEFT OUTER JOIN	TAB1을 기준으로 키 값 조합이 B-B, C-C, D-NULL, E-NULL 인 4건이 출력됩니다.
RIGHT OUTER JOIN	TAB2를 기준으로 키 값 조합이 NULL-A, B-B, C-C인 3건이 출력됩니다.
FULL OUTER JOIN	- 양쪽 테이블이 기준이되며 키 값 조합이 NULL-A, B-B, C-C, D-NULL, E-NULL 인 5건이 출력됩니다. - LEFT JOIN과 RIGHT JOIN의 합집합과 같음을 확인할 수 있습니다.
CROSS JOIN	양쪽 테이블 TAB1과 TAB2의 데이터를 곱한 개수인 12건이 출력됩니다.

▼ 연습문제

☑️ 문제 1

Q. 문제	합집합을 만들 때 정렬을 유발하는 것은?
A. (1)	EXCEPT
A. (2)	MINUS
A. (3)	UNION ALL

A. (4)	UNION

▼ 정답

(4) UNION 연산은 2개의 테이블을 하나로 합치며 중복 데이터를 제거한다. 이에 UNION은 정렬 과정을 발생시킨다

☑️ 문제 2

Q. 문제	Oracle 데이터베이스에서 차집합을 구하는 것은?
A. (1)	INTERSECTION
A. (2)	UNION ALL
A. (3)	MINUS
A. (4)	DIVIDE

▼ 정답

(3) MINUS

☑️ 문제 3

Q. 문제	데이터의 행 수가 가장 많이 조회되는 것은?
A. (1)	LEFT OUTER JOIN
A. (2)	RIGHT OUTER JOIN
A. (3)	CORSS JOIN
A. (4)	INNER JOIN

▼ 정답

(3) CROSS JOIN은 카테시안 곱이 발생하여 많은 행이 조회된다

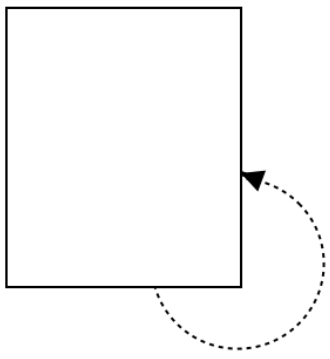
03. 계층형 질의와 표준 셀프 조인

✔️ **계층형 데이터를 조회하기 위해 사용하는 계층형 질의의 대해 학습합니다.**

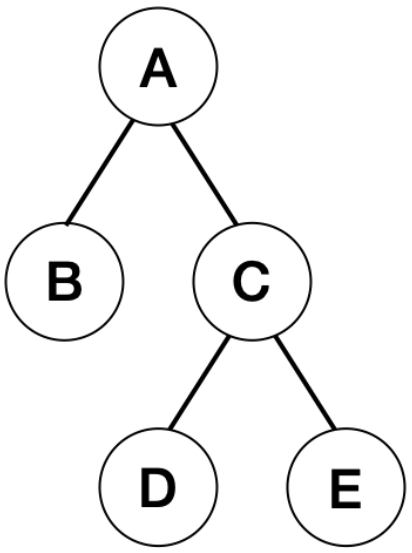
▼ 1) 계층형 질의란?

계층형 질의를 이해하기 위해서는 계층형 데이터에 대한 이해가 선행되어야 합니다. 계층형 데이터는 동일 테이블에 계층적으로 상위와 하위 데이터가 포함된 데이터를 의미합니다. 예를 들어, 회사의 상위 부서와 그 아래에 있는 하위 부서의 구조를 떠올려보면 상 → 하 구조로 이루어져 있는 것을 확인할 수 있는데, 이런 데이터가 바로 계층형 데이터입니다. 현실 세계의 정보는 계층적으로 구성된 경우가 많습니다. 학교는 선생님과 학생, 회사는 사원과 사장 등의 관계가 모두 계층적인 관계라고 할 수 있습니다.

이러한 계층형 구조는 데이터 모델 설계를 통해 테이블로 만들 수 있습니다. 아래 구조를 살펴보면 A 사원은 부모 데이터 즉, 관리자가 없음을 알 수 있고 B와 C는 모두 A 사원이 관리자임을 알 수 있습니다. 마찬가지로 D와 E는 모두 C 사원이 관리자임을 계층적 구조와 샘플 데이터 테이블을 통해 파악할 수 있습니다.



순환관계 데이터 모델



계층적 구조

사원	관리자
A	
B	A
C	A
D	C
E	C

샘플 데이터

트리 구조(그림)로 보면 쉽지만, 테이블로 계층형을 보기는 어렵다

엔터티를 순환관계 데이터 모델로 설계할 경우도 계층형 데이터가 발생합니다. 이러한 계층형 데이터를 조회하기 위해서는 계층형 질의를 사용해야 합니다. 이런 계층형 질의는 DBMS를 제공하는 회사 별로 구조적 차이가 있습니다.

▼ *참고 - 간단히 알아보는 용어 정리

- LEVEL - 각 데이터의 계층
- NODE - 각각의 데이터
- ROOT (NODE) - 최상위 노드
- Parent Node - 노드의 상위 노드
- Child Node - 노드의 하위 노드
- Leaf Node - 하위 노드가 없는 노드

▼ 2) Oracle 계층형 질의

Oracle에서 계층형 질의를 하기 위한 문법은 다음과 같습니다.

a. Oracle 계층형 질의 구문

- **SELECT 칼럼**
  - 조회하고자 하는 칼럼을 지정합니다.
- **FROM 테이블**
  - 대상이 되는 테이블을 지정합니다.
- **WHERE**
  - 모든 전개를 수행하고 나서 지정 조건을 만족하는 데이터만 추출할 수 있습니다.
  - 계층적으로 진행 후에 특정한 조건에 맞는 데이터만 가져옵니다.
- **START WITH**
  - 계층 구조 전개의 시작 위치를 지정하는 구문입니다.
  - 어디서부터 계층 질의를 시작하는지 루트 데이터(루트 노드 행)를 설정하는 구문입니다.
- **CONNECT BY [NOCYCLE]**
  - 전개되어질 자식 데이터를 지정하는 구문 (연결 고리)
  - 자식 데이터는 CONNECT BY 절에 주어진 조건을 만족해야 합니다.
  - NOCYCLE

        - 데이터를 전개하면서 "이미 나타났던 동일한 데이터가 전개 중에 다시 나타나는 경우를 가리켜 사이클이 형성되었다"라고 합니다.
        - 이런 사이클이 발생한 데이터는 오류가 발생하는데 NOCYCLE 구문을 추가하면 사이클이 발생한 이후의 데이터는 전개하지 않게 할 수 있습니다.
- [**PRIOR] A AND B**
    - CONNECT BY 절에 사용되며, 현재 읽은 칼럼을 지정할 수 있습니다.
    - `PRIOR 자식 = 부모` 형태로 사용
        - 부모 데이터에서 자식 데이터 (부모 → 자식) 방향으로 작성하면 순방향
            - `PRIOR 사원 = 관리자`
        - 자식 데이터에서 부모 데이터 (자식 → 부모) 방향으로 작성하면 역방향
            - `PRIOR 관리자 = 사원`

      → 순환 구조를 만든다 == 계층(LEVEL)을 만들어준다

      → 이전의(prior) 컬럼 A가 = 현재의 컬럼 B와 같으면 이전에 가지고 있던 LEVEL에 1을 더해서 하위 LEVEL로 지정
    - 계층형 질의에서 문제가 나온다면 🤔 `START WITH` & `PRIOR` 부분을 해석하는게 핵심!

- **ORDER SIBLINGS BY**
    - 형제 노드(동일한 LEVEL) 사이에서 정렬을 수행하는 구문입니다.

b. **계층형 질의를 사용할 때 다음과 같은 가상 칼럼(Pseudo Column)을 제공합니다.**

- **LEVEL**
    - 루트 데이터이면 1이 값으로 정해지고 그 하위 데이터이면 2가 값으로 정해집니다.
    - 하위 데이터로 내려갈수록 1씩 증가하게 됩니다.
- **CONNECT_BY_ISLEAF**
    - 전개 과정에서 해당 데이터가 각 트리 경로의 마지막 값이라면 1이되고 그렇지 않다면 0이 출력됩니다.
- **CONNECT_BY_ISCYCLE**
    - 전개 과정에서 자식 데이터를 확인합니다.
    - 그 자식 데이터를 전개했을 때 다시 부모 데이터가 정해져 있다면 무한 반복으로 전개가 될 것입니다. 이렇게 무한 반복이 되는 경우 값이 1, 그렇지 않으면 값이 0이 됩니다.
    - 무한 반복을 방지하기 위한 NOCYCLE 옵션을 사용했을 때만 사용할 수 있습니다.

- 기본 구조

```
SELECT COL,...[LEVEL, CONNECT_BY_ISLEAF, CONNECT_BY_ISCYCLE]
FROM TB1
WHERE condition AND condition
START WITH condition
CONNECT BY [NOCYCLE] condition AND condition ...
[ORDER SIBLINGS BY COL1, COL2, ...]
```

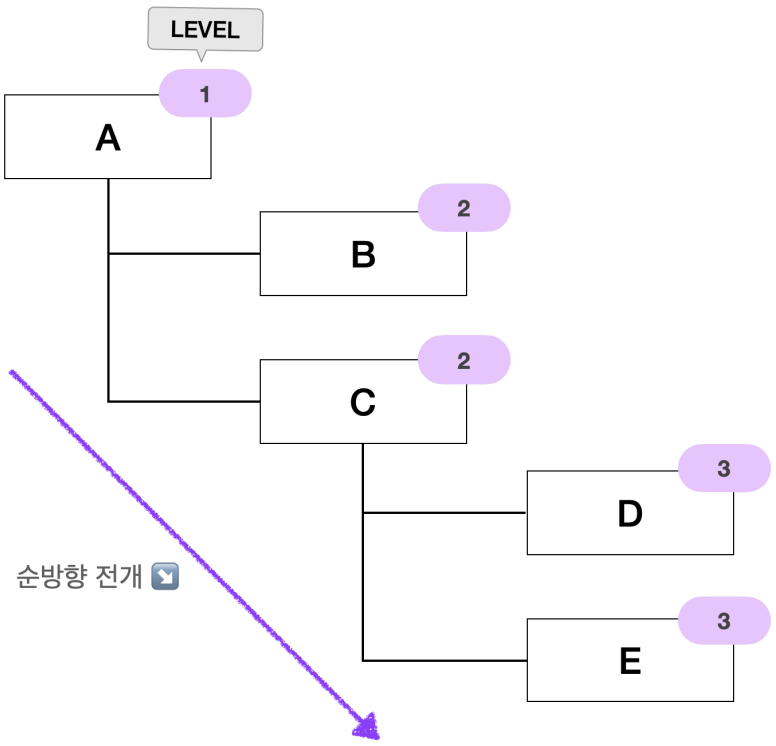
- 순방향 예시
    - 다음은 mgr → emp 방향으로 계층형 데이터를 순방향 전개해 나가는 SQL 문입니다.

❕참조 - LPAD

`LPAD("값", "문자의 총 길이", "채울 문자")`

지정한 길이만큼 왼쪽부터 특정한 문자로 채워줍니다.

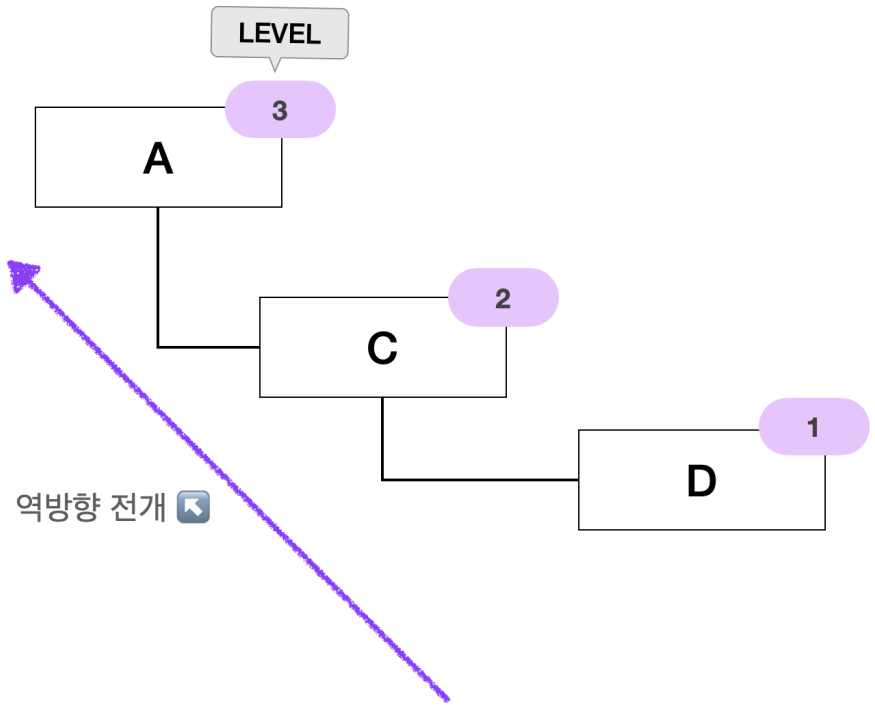
```
SELECT LEVEL,
       -- 값이 없을땐 null 출력됨 그래서 || empno 을 이용하여 현재 empno정보를 출력
       LPAD(' ', 4 * (LEVEL-1)) || empno AS EMPNO,
       mgr,
       CONNECT_BY_ISLEAF AS ISLEAF
FROM emp
START WITH mgr IS NULL
CONNECT BY PRIOR empno = mgr;
```



	"LEVEL"	EMPNO	MGR	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

- 역방향 예시
  - 다음은 emp → mgr 방향의 역방향 전개해 나가는 SQL문입니다.

```
SELECT
       LEVEL,
       LPAD(' ', 4 * (LEVEL - 1)) || empno AS EMPNO,
       mgr,
       CONNECT_BY_IS_LEAF AS ISLEAF
FROM emp
START WITH empno = 7876
CONNECT BY PRIOR mgr = empno;
```



	"LEVEL"	EMPNO	MGR	ISLEAF
1	1	7876	7788	0
2	2	7788	7566	0
3	3	7566	7839	0
4	4	7839	<null>	1

- 실습

```
-- 테이블 생성
CREATE TABLE organization (
    id NUMBER PRIMARY KEY,
    name VARCHAR2(50),
    parent_id NUMBER
);

-- 데이터 입력
INSERT INTO organization (id, name, parent_id)
VALUES (1, 'CEO', NULL);

INSERT INTO organization (id, name, parent_id)
VALUES (2, 'HR Manager', 1);

INSERT INTO organization (id, name, parent_id)
VALUES (3, 'Finance Manager', 1);

INSERT INTO organization (id, name, parent_id)
VALUES (4, 'Employee 1', 2);

INSERT INTO organization (id, name, parent_id)
VALUES (5, 'Employee 2', 2);

INSERT INTO organization (id, name, parent_id)
VALUES (6, 'Employee 3', 3);
```

```
SELECT id 부서번호, name 부서이름, parent_id 상위부서번호 FROM organization;
```

부서번호	부서이름	상위부서번호
1	CEO	-
2	HR Manager	1
3	Finance Manager	1
4	Employee 1	2
5	Employee 2	2
6	Employee 3	3

```
-- 계층형 쿼리 (순방향)

SELECT id 부서번호, name 부서이름, parent_id 상위부서번호,
    LPAD(' ', (level - 1) * 4) || name AS 계층도
FROM organization
START WITH parent_id IS NULL
CONNECT BY PRIOR id = parent_id
ORDER BY level;
```

부서번호	부서이름	상위부서번호	계층도
1	CEO	-	CEO
2	HR Manager	1	HR Manager
3	Finance Manager	1	Finance Manager
4	Employee 1	2	Employee 1
5	Employee 2	2	Employee 2
6	Employee 3	3	Employee 3

```
-- 계층형 쿼리 (역방향)

SELECT id 부서번호, name 부서이름, parent_id 상위부서번호,
    LPAD(' ', (level - 1) * 4) || name AS 계층도
FROM organization
START WITH parent_id = 3
CONNECT BY PRIOR parent_id = id
ORDER BY level;
```

부서번호	부서이름	상위부서번호	계층도
6	Employee 3	3	Employee 3
3	Finance Manager	1	Finance Manager
1	CEO	-	CEO

c. (참고) Oracle은 계층형 질의를 사용할 때 사용자 편의성을 위해 추가적인 함수를 제공합니다.

- **SYS_CONNECT_BY_PATH(칼럼명, 경로구분자)**
  - 루트 데이터로부터 현재 전개할 데이터까지의 경로를 표시해주는 함수입니다.
- **CONNECT_BY_ROOT 칼럼**
  - 현재 전개할 데이터의 루트 데이터를 표시하는 단일행 연산자 입니다.
- **예시**
  - START WITH 로 시작하는 루트 데이터가 1건이기 때문에 CONNECT_BY_ROOT empno 는 전부 1개의 값으로 출력됩니다.
  - SYS_CONNECT_BY_PATH 는 현재 데이터까지의 경로이기 때문에 값이 D라면 A / C / D 로 출력이 됩니다.

```
SELECT
    CONNECT_BY_ROOT empno AS ROOT_EMPNO,
    SYS_CONNECT_BY_PATH(empno, '/') AS 경로,
    empno,
```

```
    mgr
FROM emp
START WITH mgr IS NULL
CONNECT BY PRIOR empno = mgr;
```

ROOT_EMPNO	경로	EMPNO	MGR
7839	/7839	7839	<null>
7839	/7839/7566	7566	7839
7839	/7839/7566/7788	7788	7566
7839	/7839/7566/7788/7876	7876	7788
7839	/7839/7566/7902	7902	7566
7839	/7839/7566/7902/7369	7369	7902
7839	/7839/7698	7698	7839
7839	/7839/7698/7499	7499	7698
7839	/7839/7698/7521	7521	7698
7839	/7839/7698/7654	7654	7698
7839	/7839/7698/7844	7844	7698

- 추가 실습

▼ 실습 데이터

```
CREATE TABLE sport_type (
    sport_id NUMBER PRIMARY KEY,
    sport_name VARCHAR2(255),
    main_event_id NUMBER
);

-- 스포츠 종목 데이터 삽입
INSERT INTO sport_type VALUES (1, '축구', NULL);
INSERT INTO sport_type VALUES (2, '농구', NULL);
INSERT INTO sport_type VALUES (3, '수영', NULL);
INSERT INTO sport_type VALUES (4, '테니스', NULL);
INSERT INTO sport_type VALUES (5, '육상', NULL);

-- 스포츠 종목의 하위 종목 데이터 삽입
INSERT INTO sport_type VALUES (11, '축구 - 월드컵', 1);
INSERT INTO sport_type VALUES (12, '축구 - 유소년 리그', 1);
INSERT INTO sport_type VALUES (21, '농구 - NBA', 2);
INSERT INTO sport_type VALUES (22, '농구 - 대학 리그', 2);
INSERT INTO sport_type VALUES (31, '수영 - 올림픽', 3);
INSERT INTO sport_type VALUES (32, '수영 - 대회', 3);
```

> sport_type 테이블의 현재 스포츠 종목 ID와 이름을 출력하고 최상위 종목 ID와 하위 종목이 없는 마지막 종목인지 확인해주세요. 그리고 추가로 sport 종목의 경로도 구분자를 `/` 로 해서 같이 출력해보세요.

```
SELECT
       sport_id,
       sport_name,
       CONNECT_BY_ROOT sport_id AS "ROOT SPORT",
       CONNECT_BY_ISLEAF AS "IS LEAF",
       SYS_CONNECT_BY_PATH(sport_id, '/')
FROM sport_type
START WITH main_event_id IS NULL
CONNECT BY PRIOR sport_id = main_event_id;
```

▼ 3) (참고) SQL Server 계층형 질의

과거 SQL Server는 계층형 질의를 하기 위해 복잡한 과정이 필요했습니다. 하지만 지금은 예전보다 훨씬 더 간편한 형태로 계층형 질의를 수행할 수 있습니다.

우선 CTE(Common Table Expression) 을 재귀 호출해야 하는데, CTE는 공통 테이블 식으로도 불리며, 임시로 이름이 지정된 결과 집합을 의미합니다. CTE의 기본 구조는 다음과 같습니다.

- 기본 구조

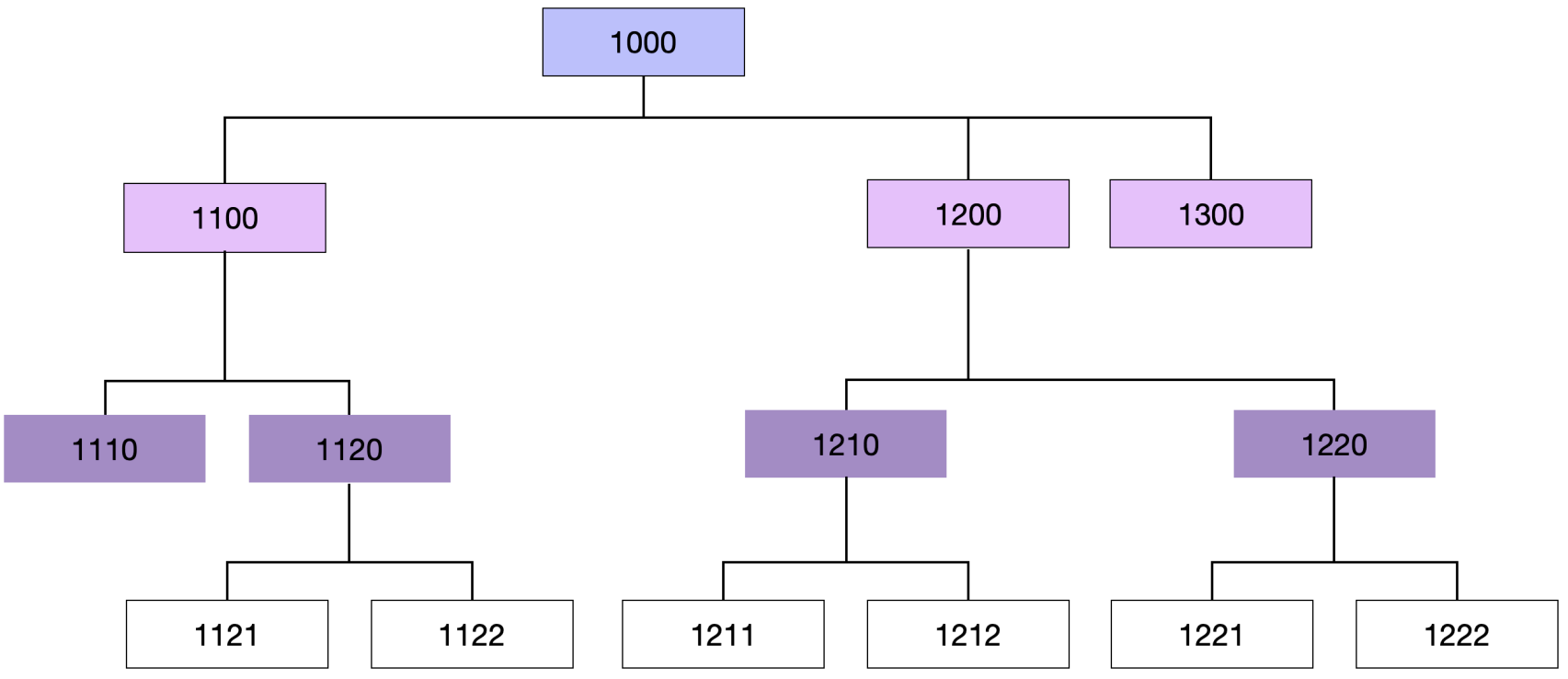
```
WITH expression_name [ (칼럼명[ ,...n ] ) ]
    AS
    ( CTE_query_definition ) [ ,...n ] ]
```

- 예시

```
WITH employee_anchor
  AS
  ( SELECT employee_id,
         last_name,
         first_name,
         report_to,
         0 AS LEVEL
  FROM employees
  WHERE report_to IS NULL -- 재귀 호출의 시작점
  UNION ALL
  SELECT r.employeed_id,
         r.last_name,
         r.first_name,
         r.report_to,
         a.LEVEL + 1
  FROM employee_anchor a, employee r
  WHERE a.employee_id = r.repost_to )


SELECT LEVEL,
       employee_id,
       last_name,
       first_name,
       report_to
FROM employee_anchor GO;
```

  - CTE_query_definition 으로 정의된 곳을 보면 현재 UNION ALL 연산자로 두 개의 쿼리를 결합한 상태입니다. 둘 중 위에 있는 쿼리를 '앵커 맴버(Anchor Member)' 라고 하고 아래에 있는 쿼리는 '재귀 맴버(Recursive Member)'라고 부릅니다.
  - CTE 재귀 호출로 만들어낸 계층 구조의 결과는 실제와는 다른 모습으로 출력이 됩니다. 따라서 조직도와 같은 모습으로 출력하려면 ORDER BY 절을 추가하여 정렬을 해야 합니다.

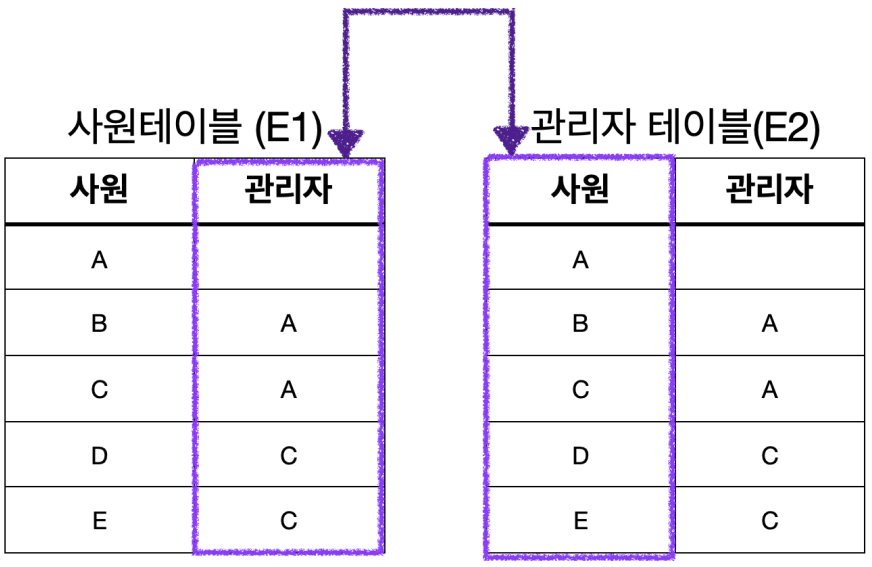


💡 **재귀적 쿼리 처리 과정**

1. CTE 식을 앵커맴버와 재귀맴버로 분할합니다.
2. 앵커 맴버를 실행하여 첫 번째 호출 또는 기본 결과 집합(T_0)을 만들어 줍니다.
3. T_i 는 입력으로 사용하고 $T_i + 1$은 출력으로 사용하여 재귀 맴버를 실행해 줍니다.
4. 빈 집합이 반환될 때까지 위의 3단계를 반복합니다.
5. T_0 ~ T_n까지의 UNION ALL의 결과 집합을 반환합니다.

▼ 4) 셀프 조인(SELF JOIN)

- 셀프 조인(SELF JOIN)은 동일한 테이블 간의 JOIN을 의미합니다. FROM 절에서 동일 테이블을 두 번 이상 JOIN 테이블로 명시하면 됩니다.
- SELF JOIN을 수행하면 테이블과 칼럼의 이름이 모두 동일하기 때문에 각각의 테이블을 식별하기 위해서 반드시 테이블의 ALIAS(별칭)을 붙여주어야 합니다. 칼럼 또한 어떤 테이블의 칼럼을 의미하는지 앞에 명시해줘야 합니다.
- 나머지는 기본 JOIN 사용법과 동일합니다.



- 실습

▼ 실습 데이터 sport_type

```
CREATE TABLE sport_type (
    sport_id NUMBER PRIMARY KEY,
    sport_name VARCHAR2(255),
    main_event_id NUMBER
);

-- 스포츠 종목 데이터 삽입
INSERT INTO sport_type VALUES (1, '축구', NULL);
INSERT INTO sport_type VALUES (2, '농구', NULL);
INSERT INTO sport_type VALUES (3, '수영', NULL);
INSERT INTO sport_type VALUES (4, '테니스', NULL);
INSERT INTO sport_type VALUES (5, '육상', NULL);

-- 스포츠 종목의 하위 종목 데이터 삽입
INSERT INTO sport_type VALUES (11, '축구 - 월드컵', 1);
INSERT INTO sport_type VALUES (12, '축구 - 유소년 리그', 1);
INSERT INTO sport_type VALUES (21, '농구 - NBA', 2);
INSERT INTO sport_type VALUES (22, '농구 - 대학 리그', 2);
INSERT INTO sport_type VALUES (31, '수영 - 올림픽', 3);
INSERT INTO sport_type VALUES (32, '수영 - 대회', 3);
```

> SPORT 테이블의 각 종목별 상위 종목의 ID(UPPER ID)와 이름(UPPER NAME)을 LEFT JOIN을 이용하여 출력해보세요.

```
SELECT
       s1.sport_id,
       s1.sport_name,
       s2.sport_id "UPPER ID",
       s2.sport_name "UPPER SPORT"
FROM sport_type s1 LEFT JOIN sport_type s2
  ON s1.main_event_id = s2.sport_id
ORDER BY s1.sport_id;
```

▼ 연습문제

☑️ 문제 1

Q. 문제	**계층형 질의에 대한 설명으로 올바르지 않는 것은?**
A. (1)	START WITH절은 계층 구조의 시작점을 지정하는 구문이다.
A. (2)	순방향 전개 란 부모노드로부터 자식 노드 방향으로 전개하는 것을 말한다
A. (3)	ORDER SIBLINGS BY는 형제 노드 사이에서 정렬을 지정하는 구문이다
A. (4)	루트 노드의 LEVEL값은 0부터 시작한다.

▼ 정답

(4) 루트 노드의 LEVEL값은 1부터 시작한다.

☑️ 문제 2

Q. 문제	**계층형 데이터베이스에서 계층 구조를 쿼리하기 위한 SQL 구문 중 하나는?**
A. (1)	ORDER BY level DESC
A. (2)	JOIN parent_table ON child_table.parent_id = parent_table.parent_id
A. (3)	START WITH root_id CONNECT BY PRIOR node_id = parent_id
A. (4)	SELECT * FROM table_name

▼ 정답

(3) 계층형 데이터를 쿼리할 때, START WITH와 CONNECT BY PRIOR 구문을 사용하여 시작 노드를 지정하고 부모-자식 관계를 설정한다.

☑️ 문제 3

Q. 문제	**CONNECT BY에 대한 설명으로 옳은 것은?**
A. (1)	계층적으로 진행 후에 특정한 조건에 맞는 데이터만 가져온다
A. (2)	자식 데이터는 CONNECT BY 절에 주어진 조건을 만족해야 한다
A. (3)	계층 구조 전개의 시작 위치를 지정하는 구문이다
A. (4)	테이블 간의 조인 연산을 수행 후 정렬하는 데 사용된다

▼ 정답

(2)