KB금융그룹 | 국민의 평생 금융파트너

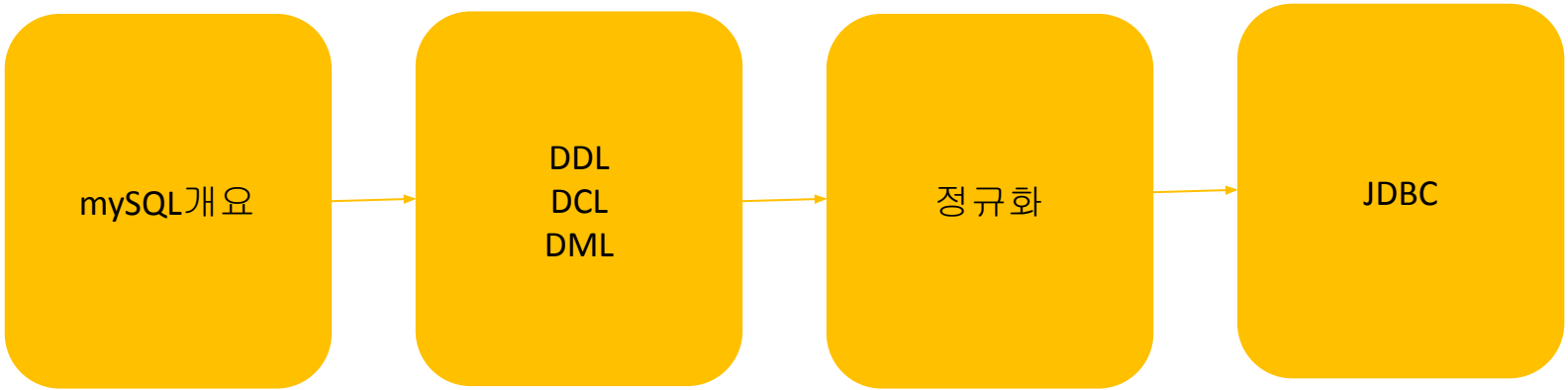
2024년 상반기 K-디지털 트레이닝

DML 심화(조인)

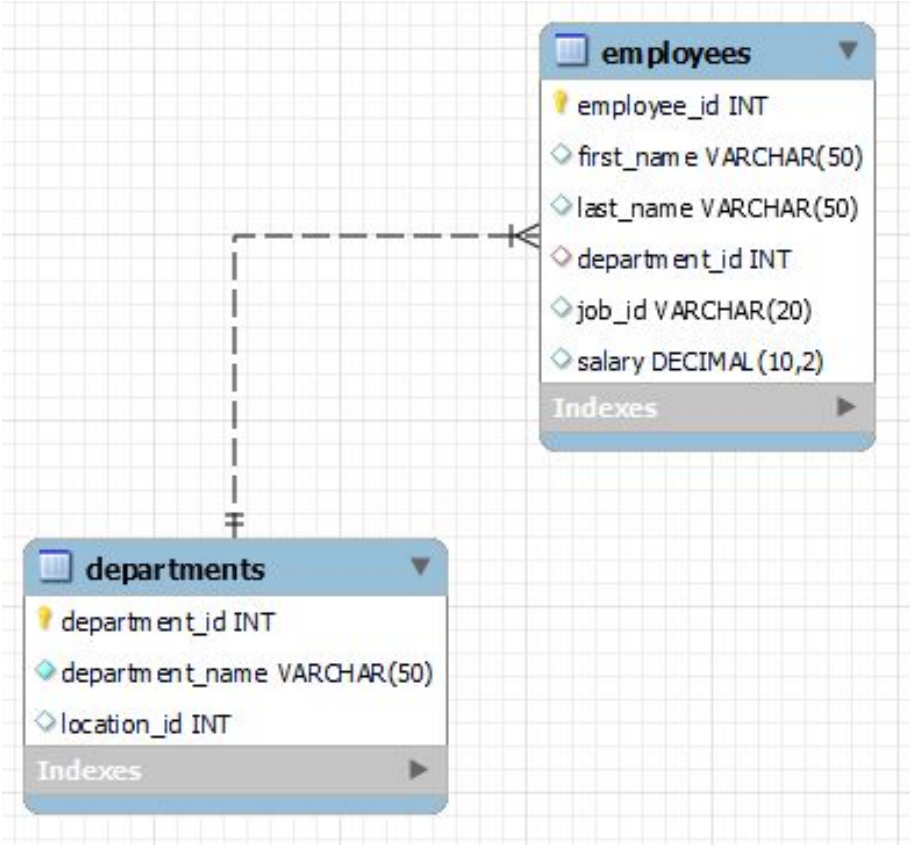
[KB] IT's Your Life

KB 국민은행

커리큘럼



학습 목표


employees
employee_id INT
first_name VARCHAR(50)
last_name VARCHAR(50)
department_id INT
job_id VARCHAR(20)
salary DECIMAL(10,2)
Indexes
departments
department_id INT
department_name VARCHAR(50)
location_id INT
Indexes

지금까지 학습내용

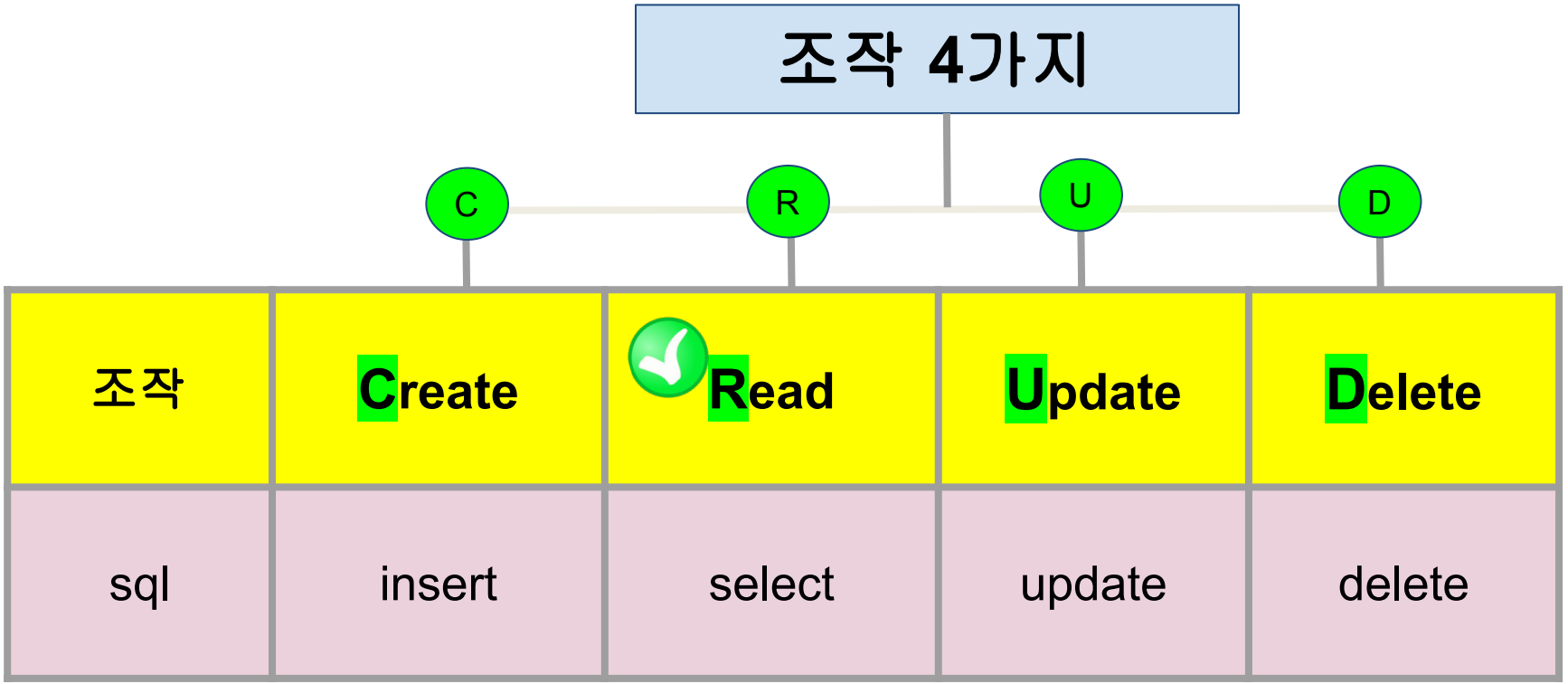
- DML
  - insert: unique, pk,fk,sysdate(), curdate(), now()
  - select:
    - where
    - is null, is not null
    - and, or, like(%, _)
    - as
    - order by, desc
    - 내장함수(문자, 숫자, 날짜, 집계/그룹)

DML 심화 개요
- 집합, 조인, 서브쿼리

SQL-명령어의 역할에 따른 분류

용어	Data Definition Language (DDL)	Data Manipulation Language (DML)	Data Control Language (DCL)	Transaction Control Language (TCL)
역할	데이터 항목 정의	데이터 조작	DBMS 제어 (계정승인, 권한부여/회수)	트랜잭션 제어
SQL 명령어	CREATE, ALTER, DROP, TRUNCATE	INSERT, SELECT, UPDATE, DELETE	GRANT, REVOKE	COMMIT, ROLLBACK

DML-CRUD, 크루드


조작 4가지
C
R
U
D
조작
Create
Read
Update
Delete
sql
insert
select
update
delete

잘못된 예제 => 제1정규화

주소

영등포구 여의도동 연락처 02-11-222

주소

영등포구 여의도동

연락처

02-11-222

테이블 생성(학생수강성적 테이블)

학생번호	학생이름	주소	학과	학과사무실	강좌이름	강의실	성적
501	박지성	멘체스타	컴퓨터과	공학관101	데이터베이스	공학관 110	3.5
401	김연아	서울	체육학과	체육관101	데이터베이스	공학관 110	4.0
402	장미란	강원도	체육학과	체육관101	스포츠경영학	체육관 103	3.5
502	추신수	클리블랜드	컴퓨터과	공학관101	자료구조	공학관 111	4.0
501	박지성	멘체스타	컴퓨터과	공학관101	자료구조	공학관 111	3.5

정규화 후 테이블

학생번호	학생이름	주소	학과
502	추신수	미국 클리블랜드	컴퓨터과
501	박지성	영국 맨체스타	컴퓨터과
402	장미란	대한민국 강원도	체육학과
401	김연아	대한민국 서울	체육학과

학생

학생번호(PK)

학생이름

주소

학과

강좌이름	강의실
데이터베이스	공학관110
스포츠경영학	체육관103
자료구조	공학관111

강좌정보

강좌이름(PK)

강의실

학생번호	강좌이름	성적
501	데이터베이스	3.5
401	데이터베이스	4.0
402	스포츠경영학	3.5
502	자료구조	4.0
501	자료구조	3.5

성적정보

학생번호(PK)

강좌이름(PK)

성적

학과	학과사무실
컴퓨터과	공학관101
체육학과	체육관101

학과정보

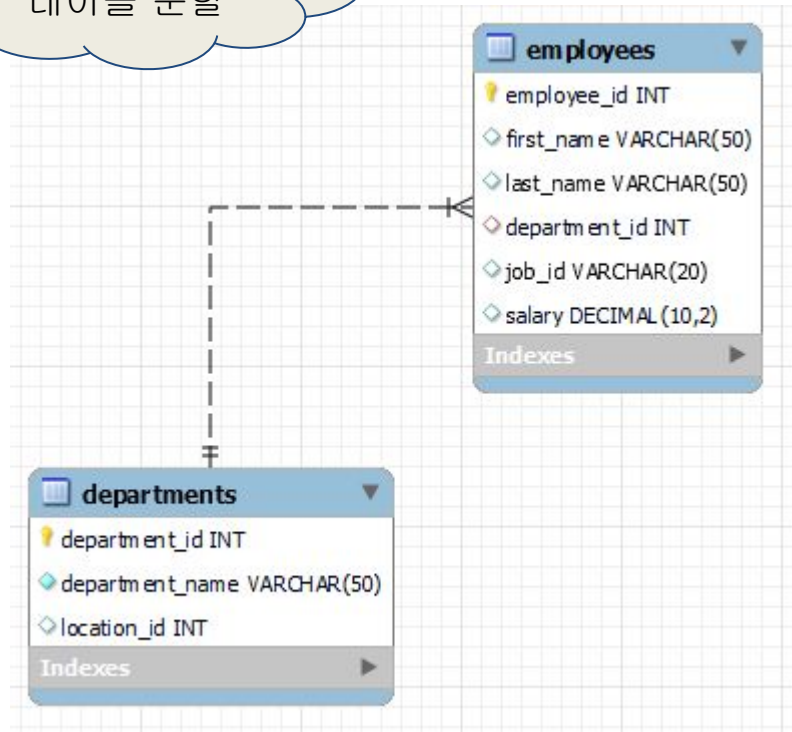
학과(PK)

학과사무실

실습예제 생성

- company
  - Tables
    - departments
      - Columns
        - department_id
        - department_name
        - location_id
      - Indexes
        - PRIMARY
      - Foreign Keys
      - Triggers
    - employees
      - Columns
        - employee_id
        - first_name
        - last_name
        - department_id
        - job_id
        - salary
      - Indexes
        - PRIMARY
        - department_id
      - Foreign Keys
        - employees_ibfk_1
      - Triggers

정규화로 인한
테이블 분할



실습예제 생성


```
-- 데이터베이스 생성
CREATE DATABASE company;
USE company;

-- departments 테이블 생성
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50) NOT NULL,
    location_id INT
);

-- employees 테이블 생성
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT,
    job_id VARCHAR(20),
    salary DECIMAL(10, 2),
    FOREIGN KEY (department_id) REFERENCES
departments(department_id)
);
```

```
-- departments 테이블에 데이터 삽입
INSERT INTO departments (department_id, department_name,
location_id) VALUES
(1, 'Sales', 1700),
(2, 'Engineering', 1800),
(3, 'HR', 1700),
(4, 'Finance', 1900);

-- employees 테이블에 데이터 삽입
INSERT INTO employees (employee_id, first_name, last_name,
department_id, job_id, salary) VALUES
(101, 'John', 'Doe', 1, 'Salesman', 60000),
(102, 'Jane', 'Smith', 2, 'Engineer', 80000),
(103, 'Sam', 'Brown', 1, 'Salesman', 62000),
(104, 'Sue', 'Wilson', 3, 'HR Specialist', 55000),
(105, 'Jim', 'Taylor', 2, 'Engineer', 78000),
(106, 'Amy', 'Adams', 4, 'Accountant', 70000);
```

실습예제 생성

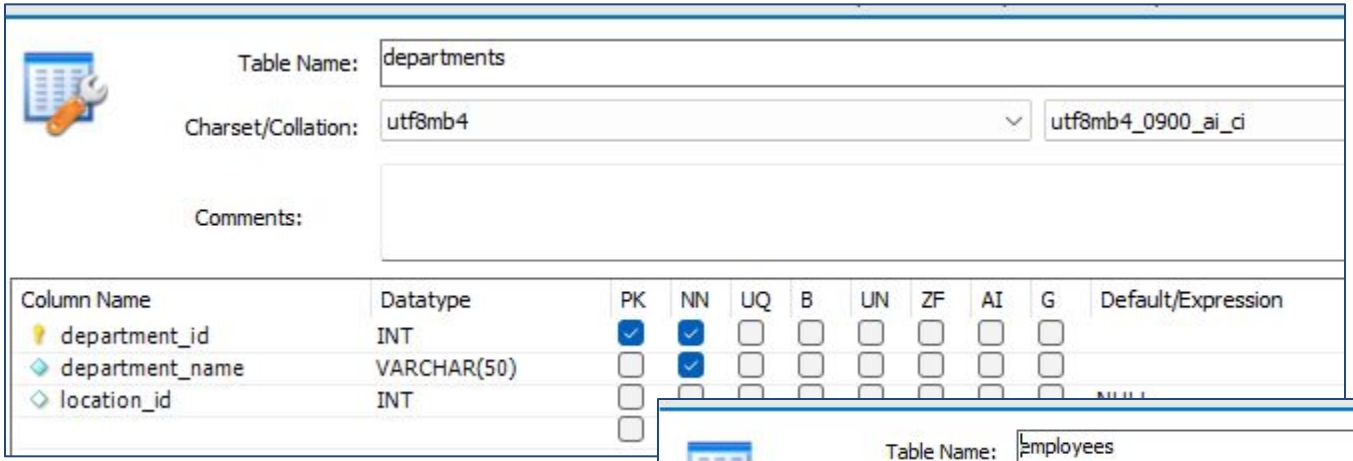


Table Name: departments
Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
Comments:
Column Name Datatype PK NN UQ B UN ZF AI G Default/Expression
department_id INT
department_name VARCHAR(50)
location_id INT

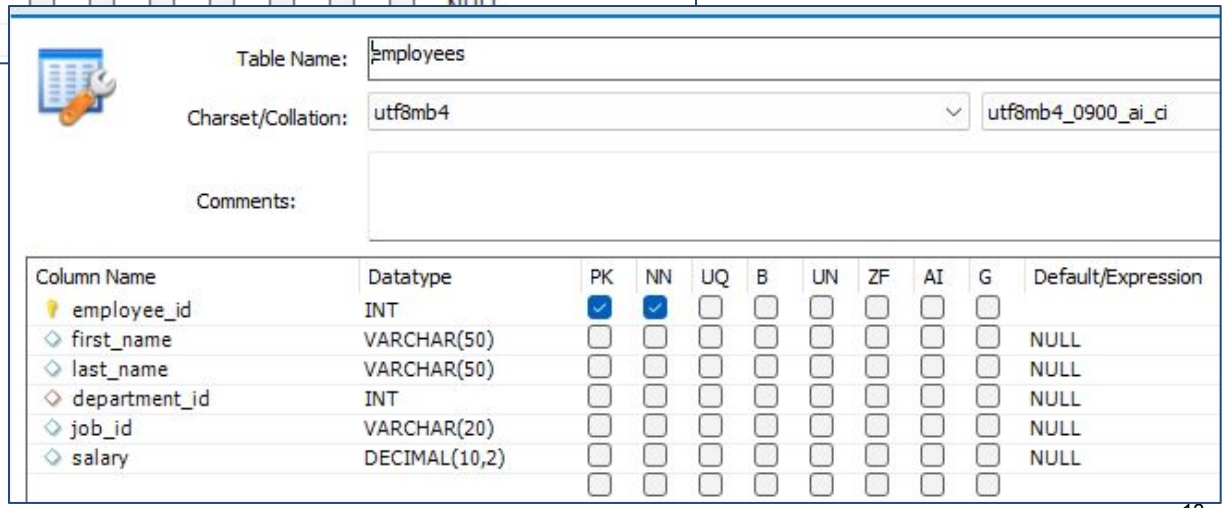


Table Name: employees
Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci
Comments:
Column Name Datatype PK NN UQ B UN ZF AI G Default/Expression
employee_id INT
first_name VARCHAR(50) NULL
last_name VARCHAR(50) NULL
department_id INT NULL
job_id VARCHAR(20) NULL
salary DECIMAL(10,2) NULL

실습예제 생성

```
1 • SELECT * FROM company.departments;
```

department_id	department_name	location_id
1	Sales	1700
2	Engineering	1800
3	HR	1700
4	Finance	1900

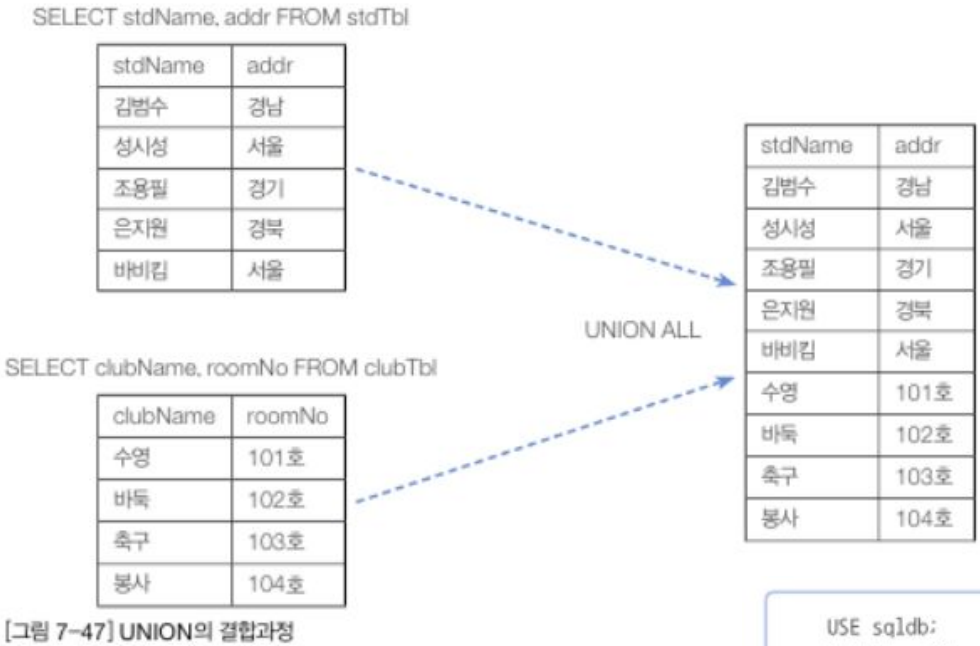
```
1 • SELECT * FROM company.employees;
```

employee_id	first_name	last_name	department_id	job_id	salary
101	John	Doe	1	Salesman	60000.00
102	Jane	Smith	2	Engineer	80000.00
103	Sam	Brown	1	Salesman	62000.00
104	Sue	Wilson	3	HR Specialist	55000.00
105	Jim	Taylor	2	Engineer	78000.00
106	Amy	Adams	4	Accountant	70000.00

집합

UNION/UNION ALL

- 열의 개수가 같고, 데이터 타입도 동일하거나 서로 호환되는 경우 **ROW** 결합이 가능



[그림 7-47] UNION의 결합과정

mysql8에서는
intersect, minus는
제공되지 않아 서브쿼리
등으로 사용해야함.

```
USE sqldb;
SELECT stdName, addr FROM stdtbl
   UNION ALL
SELECT clubName, roomNo FROM clubtbl;
```

ALL이
들어가면
중복허용

집합(set)

- MySQL 8에서 UNION과 UNION ALL 집합 연산자는 두 개 이상의 SELECT 문 결과를 결합하는 데 사용
- 두 연산자의 주요 차이점은 중복된 행 처리 방식

특성	UNION	UNION ALL
중복 행 처리	중복된 행을 제거합니다.	중복된 행을 제거하지 않습니다.
성능	일반적으로 UNION ALL보다 느립니다. 중복 제거를 위해 추가 처리 필요	UNION보다 빠릅니다. 중복 제거를 위한 추가 처리가 필요하지 않음
결과 정렬	결과는 기본적으로 오름차순으로 정렬됩니다.	결과는 정렬되지 않습니다.
사용 사례	고유한 레코드만 필요할 때	모든 레코드를 포함해야 할 때
예제	SELECT column1 FROM table1 UNION SELECT column1 FROM table2;	SELECT column1 FROM table1 UNION ALL SELECT column1 FROM table2;
결과 예시 (중복 데이터 존재 시)	중복된 행은 한 번만 표시됩니다.	중복된 행은 여러 번 표시됩니다.

집합(set)

- **UNION**
  - UNION 연산자는 두 개 이상의 SELECT 문의 결과를 결합하고 중복된 행을 제거
  - 결과 집합에서 중복된 행은 하나만 유지되며, 기본적으로 결과는 오름차순으로 정렬됨.
  - **사용 사례**: 중복된 데이터가 필요하지 않을 때 사용 예를 들어, 여러 테이블에서 고유한 레코드를 결합할 때 유용

```
-- 중복된 department_id 제거
SELECT department_id FROM employees
UNION
SELECT department_id FROM departments;
```

집합(set)

- employees 테이블과 departments 테이블에서 department_id를 선택한 결과를 결합하고 중복된 값을 제거

```
41    -- 중복된 department_id 제거
42 •  SELECT department_id FROM employees
43    UNION
44    SELECT department_id FROM departments;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

department_id
1
2
3
4

집합(set)

- **UNION ALL**
  - `UNION ALL` 연산자는 두 개 이상의 SELECT 문의 결과를 결합하지만, 중복된 행을 제거하지 않는다.
  - 모든 결과 행을 그대로 유지
  - **사용 사례**: 중복된 데이터도 모두 필요할 때 사용. 예를 들어, 두 테이블의 모든 데이터를 중복 없이 결합할 필요가 없을 때 유용

```
-- 중복된 department_id 포함
SELECT department_id FROM employees
UNION ALL
SELECT department_id FROM departments;
```

집합(set)

- employees 테이블과 departments 테이블에서 department_id를 선택한 결과를 결합하고 중복된 값을 제거하지 않는다. 모든 값이 그대로 유지

```
46    -- 중복된 department_id 포함
47 •  SELECT department_id FROM employees
48    UNION ALL
49    SELECT department_id FROM departments;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

department_id
1
1
2
2
3

집합(set)

- **UNION 사용 전제조건**
  - 조건들이 충족되지 않으면 UNION 연산자는 올바르게 작동하지 않거나 오류가 발생할 수 있음.

1) 열의 개수 일치

- 각 SELECT 문의 결과에 포함되는 **열의 개수가 동일**해야 함.
- 예

```
SELECT column1, column2 FROM table1 UNION
SELECT column3, column4 FROM table2;
```

2) 열의 데이터 타입 호환성:

- 각 SELECT 문의 결과에 포함되는 열의 데이터 타입이 서로 호환되어야 함. 반드시 같은 데이터 타입일 필요는 없지만, 서로 변환 가능해야 함.
- 예: 첫 번째 SELECT 문에서 정수를 선택하고, 두 번째 SELECT 문에서 숫자로 변환 가능한 문자열을 선택할 수 있음.

집합(set)

- **UNION 사용 전제조건**

3) 열의 순서 일치

- 각 SELECT 문의 결과에 포함되는 열의 순서가 동일해야 함.
- 첫 번째 SELECT 문의 첫 번째 열과 두 번째 SELECT 문의 첫 번째 열이 같은 의미의 데이터를 가져야 함.
- 예

```
SELECT id, name FROM table1
UNION
SELECT user_id, username FROM table2;
```

- **UNION 사용 전제조건**

4) SELECT 문의 개별 정렬 (ORDER BY) 제한

- 개별 SELECT 문에서 ORDER BY 절을 사용할 수 없음.
- 정렬은 전체 UNION 결과에 대해서만 가능
- 예

```
SELECT id, name FROM table1
UNION
SELECT id, name FROM table2
ORDER BY name; → UNION후, 정렬만 가능
```

- **UNION 사용 전제조건**

5) SELECT 문의 ALIAS 제한:

- 개별 SELECT 문에서 열에 대한 별칭(Alias)을 지정할 수 있지만, 이 별칭은 최종 결과 집합에서만 사용될 수 있음.
- 예:

```
SELECT id AS user_id, name AS username FROM table1
UNION
SELECT id, name FROM table2;
```

추가 집합 확인 문제

-- department_id를 기준으로 두 테이블의 데이터를 결합하고 중복된 행 제거

- departments 테이블의 department_id와 department_name을 선택하고, employees 테이블의 department_id와 직원의 이름을 결합하여 선택
  UNION 연산자를 사용하여 두 SELECT 문의 결과를 결합하고 중복된 행을 제거

```
-- department_id를 기준으로 두 테이블의 데이터를 결합하고 중복된 행 제거
SELECT department_id, department_name FROM departments
UNION
SELECT department_id, CONCAT(first_name, ' ', last_name) AS name FROM employees;
```

추가 집합 확인 문제

-- department_id를 기준으로 두 테이블의 데이터를 결합하고 중복된 행 제거

```
51    -- department_id를 기준으로 두 테이블의 데이터를 결합하고 중복된 행 제거
52 •  SELECT department_id, department_name FROM departments
53    UNION
54    SELECT department_id, CONCAT(first_name, ' ', last_name) AS name FROM employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

department_id	department_name
1	Sales
2	Engineering
3	HR
4	Finance
1	John Doe
2	Jane Smith
1	Sam Brown
3	Sue Wilson
2	Jim Taylor
4	Amy Adams

조인

조인 (mysql link)

조인은 기준이 있어야 함!

학생 개인 정보 테이블 + 학생 수강 신청 테이블 → 학생 (개인 정보 + 수강 신청) 테이블

‼ 집합은 출력 결과를 순서대로 붙인 것

영화가 장르별로 테이블이 만들어져 있는 상태라고 가정하자. 드라마, 스릴러, 액션 테이블 3개 전체 영화 목록을 알고자 하는 경우 집합을 사용할까? 조인을 사용할까?

조인

EMP | *<xe> Script-34 | BBS | MEMBER

Properties | Data | 엔티티 관계도

MEMBER | Enter a SQL expression to filter results (use Ctrl+Space)

그리드 / 텍스트

	ID	PW	NAME	TEL
1	apple	apple	apple	apple
2	ice	ice	ice	ice
3	kiwi	kiwi	kiwi	kiwi

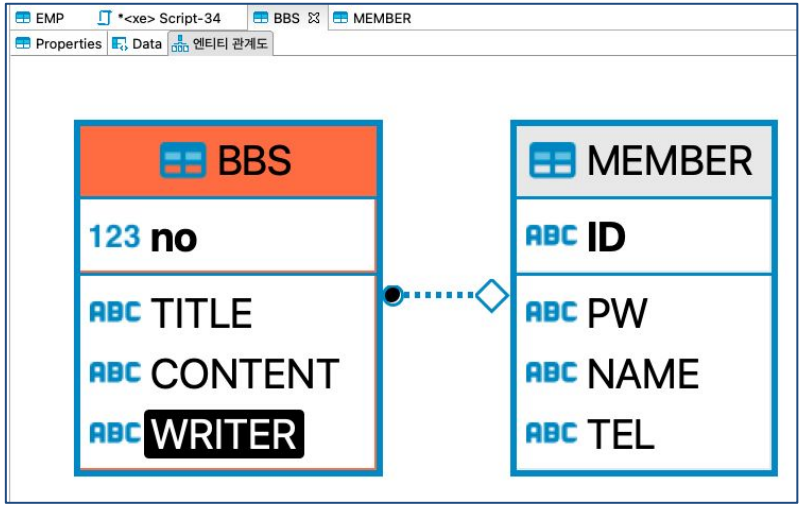
EMP | *<xe> Script-34 | BBS | MEMBER

Properties | Data | 엔티티 관계도

Table: BBS
Schema: HR
Connection: xe

Enter a SQL expression to filter results (use Ctrl+Space)

	NO	TITLE	CONTENT	WRITER
1	1	happy	happy day	ice
2	2	happy2	happy day2	ice
3	3	soso	soso day	apple
4	4	soso2	soso day2	apple



조인

- 특정 컬럼을 기준으로 테이블 컬럼들을 합함.(inner join → 내부조인)

```
01  SELECT *
02    FROM EMP, DEPT
03   WHERE EMP.DEPTNO = DEPT.DEPTNO
04   ORDER BY EMPNO;
```

합하고 싶은 테이블을 나열

기준이 되는 컬럼을 명시

:: 결과 화면

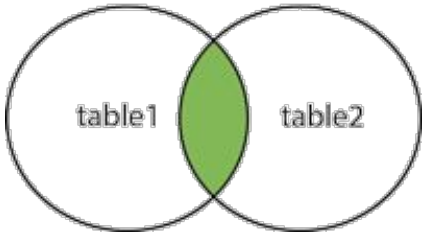
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO_1	DNAME	LOC
7369	SMITH	CLERK	7902	1980-12-17	800		20	20	RESEARCH	DALLAS
7499	ALLEN				1600	300	30	30	SALES	CHICAGO
					1250	500	30	30	SALES	CHICAGO
							20	20	RESEARCH	DALLAS
						1400	30	30	SALES	CHICAGO
							30	30	SALES	CHICAGO
							10	10	ACCOUNTING	NEW YORK
							20	20	RESEARCH	DALLAS
					3000		10	10	ACCOUNTING	NEW YORK
					1500	0	30	30	SALES	CHICAGO
				83/23	1100		20	20	RESEARCH	DALLAS
7900	JAMES		7698	1981/12/03	950		30	30	SALES	CHICAGO
7902	FORD	ANALYST	7566	1981/12/03	3000		20	20	RESEARCH	DALLAS
7934	MILLER	CLERK	7782	1982/01/23	1300		10	10	ACCOUNTING	NEW YORK

원하는 일부 컬럼만 추출하자

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.MGR, E.HIREDATE, E.SAL, E.COMM, E.DEPTNO,
       D.DNAME, D.LOC
  FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
ORDER BY EMPNO;
```

select *
from EMP e
join DEPT d
on e.deptno = d.deptno
where 조건; 도 가능

INNER JOIN

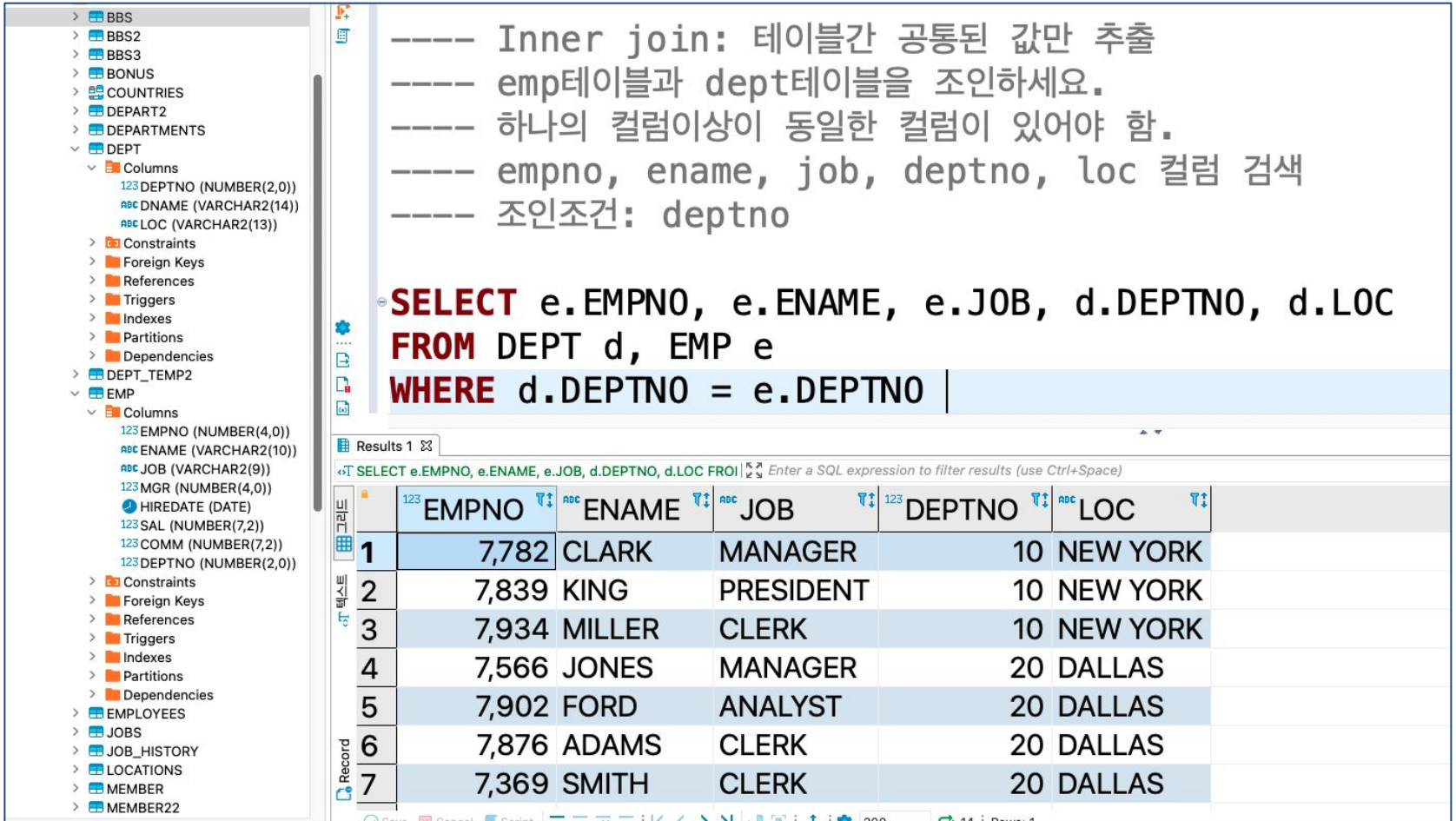


별명(alias, 닉네임)!!

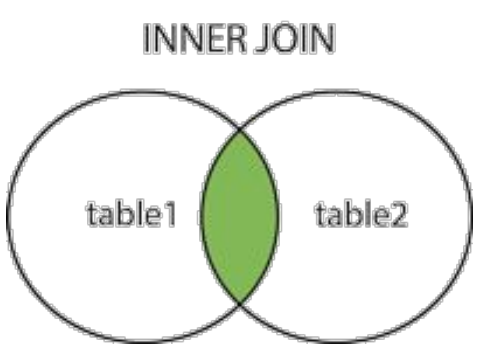
→ 더 간단해졌다.

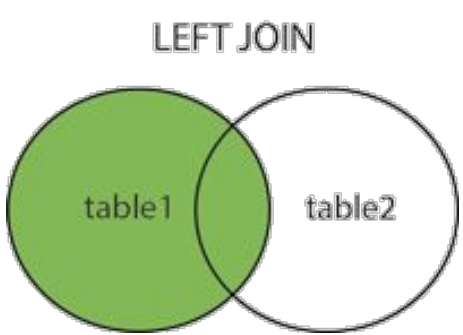
→ **EMP** 쓸 자리에 **E**로 다 쓰자

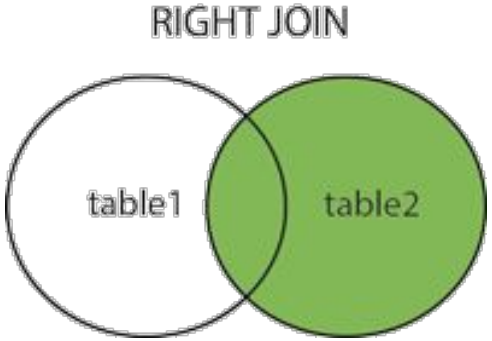
조인


BBS
BBS2
BBS3
BONUS
COUNTRIES
DEPART2
DEPARTMENTS
DEPT
Columns
DEPTNO (NUMBER(2,0))
DNAME (VARCHAR2(14))
LOC (VARCHAR2(13))
Constraints
Foreign Keys
References
Triggers
Indexes
Partitions
Dependencies
DEPT_TEMP2
EMP
Columns
EMPNO (NUMBER(4,0))
ENAME (VARCHAR2(10))
JOB (VARCHAR2(9))
MGR (NUMBER(4,0))
HIREDATE (DATE)
SAL (NUMBER(7,2))
COMM (NUMBER(7,2))
DEPTNO (NUMBER(2,0))
Constraints
Foreign Keys
References
Triggers
Indexes
Partitions
Dependencies
EMPLOYEES
JOBS
JOB_HISTORY
LOCATIONS
MEMBER
MEMBER22
---- Inner join: 테이블간 공통된 값만 추출
---- emp테이블과 dept테이블을 조인하세요.
---- 하나의 컬럼이상이 동일한 컬럼이 있어야 함.
---- empno, ename, job, deptno, loc 컬럼 검색
---- 조인조건: deptno
SELECT e.EMPNO, e.ENAME, e.JOB, d.DEPTNO, d.LOC
FROM DEPT d, EMP e
WHERE d.DEPTNO = e.DEPTNO
Results 1
SELECT e.EMPNO, e.ENAME, e.JOB, d.DEPTNO, d.LOC FROI
Enter a SQL expression to filter results (use Ctrl+Space)
그리드
텍스트
Record
EMPNO ENAME JOB DEPTNO LOC
1 7,782 CLARK MANAGER 10 NEW YORK
2 7,839 KING PRESIDENT 10 NEW YORK
3 7,934 MILLER CLERK 10 NEW YORK
4 7,566 JONES MANAGER 20 DALLAS
5 7,902 FORD ANALYST 20 DALLAS
6 7,876 ADAMS CLERK 20 DALLAS
7 7,369 SMITH CLERK 20 DALLAS

조인의 종류(oracle link, mysql link)





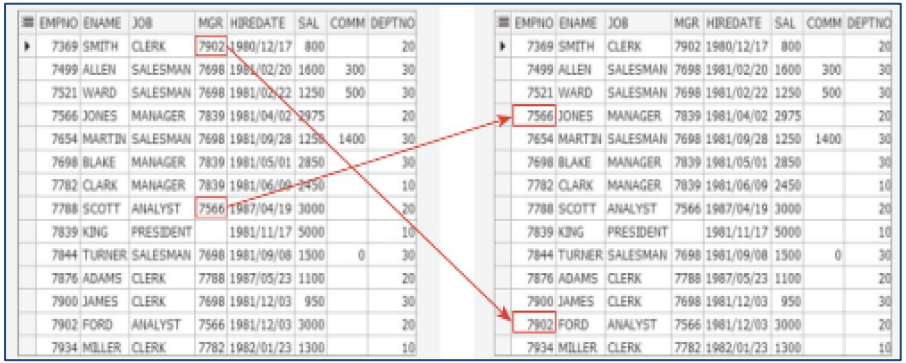


self join

```
SELECT *
  FROM EMP E, COPY_EMP C
 WHERE E.MGR = C.EMPNO;
```

검색의 결과가 해당 테이블에 있는 경우 사용

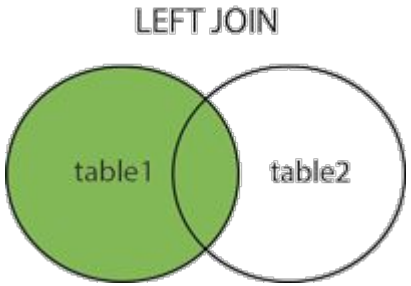
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980/12/17	800		20
7499	ALLEN	SALESMAN	7698	1981/02/20	1600	300	30
7521	WARD	SALESMAN	7698	1981/02/22	1250	500	30
7566	JONES	MANAGER	7839	1981/04/02	2975		20
7654	MARTIN	SALESMAN	7698	1981/09/28	1250	1400	30
7698	BLAKE	MANAGER	7839	1981/05/01	2850		30
7782	CLARK	MANAGER	7839	1981/06/09	2450		10
7788	SCOTT	ANALYST	7566	1987/04/19	3000		20
7839	KING	PRESIDENT		1981/11/17	5000		10
7844	TURNER	SALESMAN	7698	1981/09/08	1500	0	30
7876	ADAMS	CLERK	7788	1987/05/23	1100		20
7900	JAMES	CLERK	7698	1981/12/03	950		30
7902	FORD	ANALYST	7566	1981/12/03	3000		20
7934	MILLER	CLERK	7782	1982/01/23	1300		10



left outer join

실습 8-14 왼쪽 외부 조인을 SQL-99로 작성하기

```
01  SELECT E1.EMPNO, E1.ENAME, E1.MGR,
02         E2.EMPNO AS MGR_EMPNO,
03         E2.ENAME AS MGR_ENAME
04    FROM EMP E1 LEFT OUTER JOIN EMP E2 ON (E1.MGR = E2.EMPNO)
05  ORDER BY E1.EMPNO;
```



:: 결과 화면

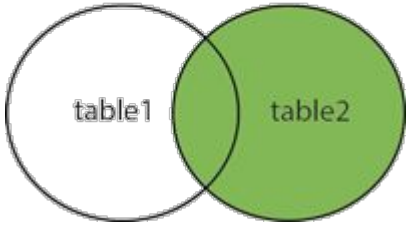
EMPNO	ENAME	MGR	MGR_EMPNO	MGR_ENAME
7369	SMITH	7902	7902	FORD
7499	ALLEN	7698	7698	BLAKE
7521	WARD	7698	7698	BLAKE
7566	JONES	7839	7839	KING
7654	MARTIN	7698	7698	BLAKE
7698	BLAKE	7839	7839	KING
7782	CLARK	7839	7839	KING
7788	SCOTT	7566	7566	JONES
7839	KING			
7844	TURNER	7698	7698	BLAKE
7876	ADAMS	7788	7788	SCOTT
7900	JAMES	7698	7698	BLAKE
7902	FORD	7566	7566	JONES
7934	MILLER	7782	7782	CLARK

right outer join

실습 8-15 오른쪽 외부 조인을 SQL-99로 작성하기

```
01  SELECT E1.EMPNO, E1.ENAME, E1.MGR,
02         E2.EMPNO AS MGR_EMPNO,
03         E2.ENAME AS MGR_ENAME
04    FROM EMP E1 RIGHT OUTER JOIN EMP E2 ON (E1.MGR = E2.EMPNO)
05  ORDER BY E1.EMPNO, MGR_EMPNO;
```

RIGHT JOIN



:: 결과 화면

EMPNO	ENAME	MGR	MGR_EMPNO	MGR_ENAME
7369	SMITH	7902	7902	FORD
7499	ALLEN	7698	7698	BLAKE
7521	WARD	7698	7698	BLAKE
7566	JONES	7839	7839	KING
7654	MARTIN	7698	7698	BLAKE
7698	BLAKE	7839	7839	KING
7782	CLARK	7839	7839	KING
7788	SCOTT	7566	7566	JONES
7844	TURNER	7698	7698	BLAKE
7876	ADAMS	7788	7788	SCOTT
7900	JAMES	7698	7698	BLAKE
7902	FORD	7566	7566	JONES
7934	MILLER	7782	7782	CLARK
			7369	SMITH
			7499	ALLEN
			7521	WARD

right outer join


```
SELECT e.EMPNO, e.ENAME , d.DEPTNO
FROM EMP e
LEFT OUTER JOIN DEPT d
ON (e.DEPTNO = d.DEPTNO)

SELECT d.DEPTNO, e.EMPNO, e.ENAME
FROM EMP e
RIGHT OUTER JOIN DEPT d
ON (e.DEPTNO = d.DEPTNO)
```

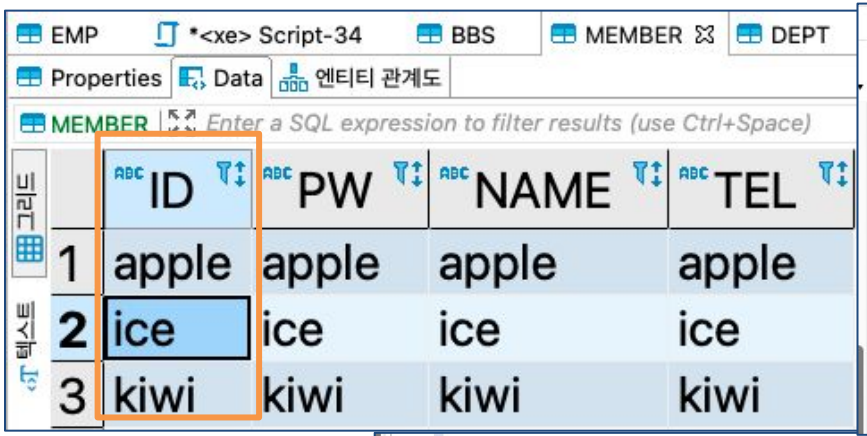
Results 1

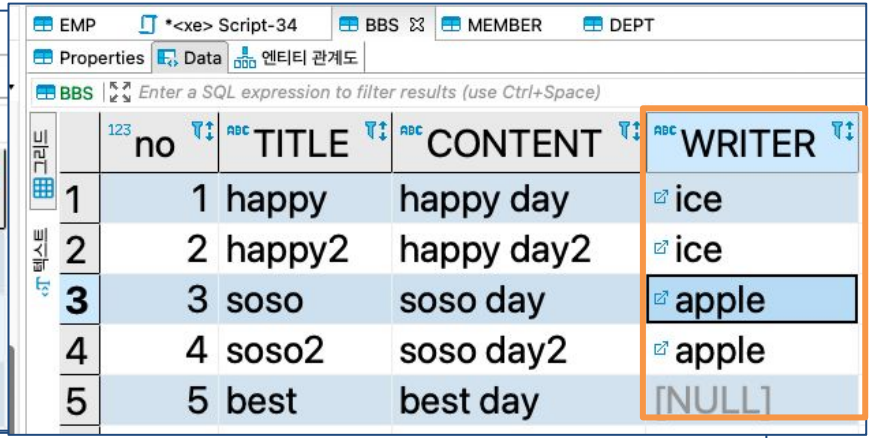
SELECT d.DEPTNO, e.EMPNO, e.ENAME FROM EMP e RIGHT OUTER JC | Enter a SQL expression to filter results (use Ct

그리드 | 텍스트

	DEPTNO	EMPNO	ENAME
15	10	100	1
16	10	200	1
17	10	300	1
18	10	400	1
19	10	500	1
20	40	[NULL]	[NULL]

inner join: 조인조건이 공통적인 것만 검색






```
SELECT *
FROM "MEMBER" m, BBS b
WHERE m.ID = b.WRITER
```

outer join

```
SELECT m.ID, m.NAME, b.TITLE, b.CONTENT
FROM "MEMBER" m
LEFT OUTER JOIN BBS b
ON (m.ID = b.WRITER)
```

Results 1

SELECT m.ID, m.NAME, b.TITLE, b.CONTENT FROM "MEMBE | Enter a SQL expression to filter results (use Ctrl+Spa

	ID	NAME	TITLE	CONTENT
1	apple	apple	soso	soso day
2	apple	apple	soso2	soso day2
3	ice	ice	happy2	happy day2
4	ice	ice	happy	happy day
5	kiwi	kiwi	[NULL]	[NULL]

```
SELECT m.ID, m.NAME, b.TITLE, b.CONTENT
FROM "MEMBER" m
RIGHT OUTER JOIN BBS b
ON (m.ID = b.WRITER)
```

Results 1

SELECT m.ID, m.NAME, b.TITLE, b.CONTENT FROM "MEMBE | Enter a SQL expression to filter results (use Ctrl+Space)

	ID	NAME	TITLE	CONTENT
1	ice	ice	happy	happy day
2	ice	ice	happy2	happy day2
3	apple	apple	soso	soso day
4	apple	apple	soso2	soso day2
5	[NULL]	[NULL]	best	best day

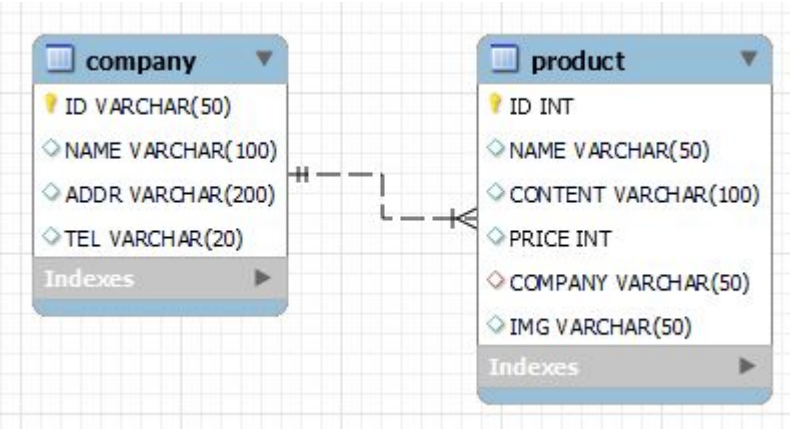
조인 확인 문제

1. inner join
2. left outer join
3. right outer join

```
CREATE TABLE COMPANY (
    ID VARCHAR(50) PRIMARY KEY,
    NAME VARCHAR(100),
    ADDR VARCHAR(200),
    TEL VARCHAR(20)
);
```

```
CREATE TABLE PRODUCT (
    ID INT PRIMARY KEY,
    NAME VARCHAR(50),
    CONTENT VARCHAR(100),
    PRICE INT,
    COMPANY VARCHAR(50),
    IMG VARCHAR(50),
    FOREIGN KEY (COMPANY) REFERENCES COMPANY(ID)
);
```

조인 확인 문제




```
INSERT INTO company (ID, NAME, ADDR, TEL) VALUES
('c100', 'good', 'seoul', '011'),
('c200', 'joa', 'busan', '012'),
('c300', 'maria', 'ulsan', '013'),
('c400', 'my', 'kwangju', '014');
```

```
INSERT INTO PRODUCT (ID, NAME, CONTENT, PRICE, COMPANY, IMG)
VALUES
(110, 'food11', 'fun food11', 11000, NULL, '11.png'),
(111, 'food12', 'fun food12', 12000, NULL, '12.png'),
(100, 'food1', 'fun food1', 1000, 'c100', '1.png'),
(101, 'food2', 'fun food2', 2000, 'c200', '2.png'),
(102, 'food3', 'fun food3', 3000, 'c300', '3.png'),
(103, 'food4', 'fun food4', 4000, 'c300', '4.png'),
(104, 'food5', 'fun food5', 5000, 'c100', '5.png'),
(105, 'food6', 'fun food6', 6000, 'c100', '6.png'),
(106, 'food7', 'fun food7', 7000, 'c200', '7.png'),
(107, 'food8', 'fun food8', 8000, 'c300', '8.png'),
(108, 'food9', 'fun food9', 9000, 'c100', '9.png'),
(109, 'food10', 'fun food10', 10000, 'c100', '10.png');
```

INNER JOIN

- 제품과 그 제품을 만든 회사의 이름을 조회하세요. 회사 정보가 없는 제품은 제외합니다.

```
SELECT P.ID AS Product_ID, P.NAME AS
Product_Name, C.NAME AS Company_Name
FROM PRODUCT P
INNER JOIN COMPANY C ON P.COMPANY = C.ID;
```

```
221 • SELECT
222       P.ID AS Product_ID,
223       P.NAME AS Product_Name,
224       C.NAME AS Company_Name
225   FROM PRODUCT P
226   INNER JOIN COMPANY C ON P.COMPANY = C.ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Product_ID	Product_Name	Company_Name
100	food1	good
104	food5	good
105	food6	good
108	food9	good
109	food10	good
101	food2	joa
106	food7	joa

LEFT OUTER JOIN

- 모든 제품과 그 제품을 만든 회사의 이름을 조회하세요. 회사 정보가 없는 제품도 포함합니다.

```
SELECT
P.ID AS Product_ID,
P.NAME AS Product_Name, C.NAME AS
Company_Name
FROM PRODUCT P
LEFT OUTER JOIN COMPANY C
ON P.COMPANY = C.ID;
```

```
234 • SELECT
235   P.ID AS Product_ID,
236   P.NAME AS Product_Name, C.NAME AS Company_Name
237   FROM PRODUCT P
238   LEFT OUTER JOIN COMPANY C
239   ON P.COMPANY = C.ID;
240
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Product_ID	Product_Name	Company_Name
105	food6	good
106	food7	joa
107	food8	maria
108	food9	good
109	food10	good
110	food11	NULL
111	food12	NULL

RIGHT OUTER JOIN

- 제품과 그 제품을 만든 회사의 이름을 조회하세요. 회사 정보가 없는 제품은 제외합니다.

```
SELECT
P.ID AS Product_ID,
P.NAME AS Product_Name, C.NAME AS
Company_Name
FROM PRODUCT P
RIGHT OUTER JOIN COMPANY C
ON P.COMPANY = C.ID;
```

```
241 •  SELECT
242    P.ID AS Product_ID,
243    P.NAME AS Product_Name, C.NAME AS Company_Name
244    FROM PRODUCT P
245    RIGHT OUTER JOIN COMPANY C
246    ON P.COMPANY = C.ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Product_ID	Product_Name	Company_Name
108	food9	good
109	food10	good
101	food2	joa
106	food7	joa
102	food3	maria
103	food4	maria
107	food8	maria
NULL	NULL	my

subquery

subquery

- SQL에서 하나의 쿼리 내에 포함된 또 다른 쿼리
- 서브쿼리는 메인 쿼리의 일부로 사용되며, 결과 집합을 메인 쿼리에서 사용하기 위해 독립적으로 실행
- 서브쿼리는 여러 가지 방식으로 메인 쿼리의 데이터를 보완하거나 필터링하는 데 사용
- **독립적 실행**
  - 서브쿼리는 메인 쿼리와 독립적으로 실행될 수 있으며, 그 결과를 메인 쿼리에서 사용
- **위치**
  - 서브쿼리는 SELECT, FROM, WHERE, HAVING, 또는 다른 SQL 절 내에 위치
- **장점**
  - 가독성 : 복잡한 쿼리를 여러 부분으로 나누어 가독성을 향상
  - 재사용성 : 서브쿼리를 통해 중복된 쿼리 논리를 재사용
  - 유연성 : 다양한 형태의 데이터를 쉽게 조작하고 필터링할 수 있음

subquery 종류

서브쿼리 종류	설명
단일 행 서브쿼리 (Single-row Subquery)	하나의 행만 반환하는 서브쿼리
다중 행 서브쿼리 (Multi-row Subquery)	여러 행을 반환하는 서브쿼리
다중 열 서브쿼리 (Multi-column Subquery)	여러 열을 반환하는 서브쿼리
상관 서브쿼리 (Correlated Subquery)	메인 쿼리의 각 행에 대해 한 번씩 실행되는 서브쿼리
스칼라 서브쿼리 (Scalar Subquery)	하나의 값만 반환하는 서브쿼리
EXISTS 서브쿼리	서브쿼리가 레코드를 반환하는지 여부를 확인하는 서브쿼리

단일 행 서브쿼리 (Single-row Subquery)

- 단일 값을 반환하는 서브쿼리로, 주로 **WHERE** 절에서 사용

```
SELECT first_name, last_name
FROM employees
WHERE department_id = (
    SELECT department_id
    FROM departments
    WHERE department_name = 'Sales'
);
```

```
57 • SELECT first_name, last_name
58   FROM employees
59   WHERE department_id = (
60       SELECT department_id
61       FROM departments
62       WHERE department_name = 'Sales'
63   );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
John	Doe
Sam	Brown

```
65 • SELECT department_id
66       FROM departments
67       WHERE department_name = 'Sales';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

department_id
1

다중 행 서브쿼리 (Multi-row Subquery)

- 여러 행을 반환하는 서브쿼리로, 주로 **IN, ANY, ALL** 키워드와 함께 사용

```
SELECT first_name, last_name
FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM departments
    WHERE location_id = 1700
);
```

```
65 •  SELECT first_name, last_name
66    FROM employees
67    WHERE department_id IN (
68        SELECT department_id
69        FROM departments
70        WHERE location_id = 1700
71    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
John	Doe
Sam	Brown
Sue	Wilson

```
73 •  SELECT department_id
74        FROM departments
75        WHERE location_id = 1700;
```

Result Grid | Filter Rows: | Edit: | Export/Import

department_id
1
3

다중 행 서브쿼리 (Multi-row Subquery)

- 어떤 부서의 직원이라도 급여가 **70,000** 이상인 직원들을 찾으세요.**(ANY)**

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary >= ANY (
    SELECT salary
    FROM employees
    WHERE salary >= 70000
);
```

```
117 •  SELECT first_name, last_name, salary
118    FROM employees
119    WHERE salary >= ANY (
120        SELECT salary
121        FROM employees
122        WHERE salary >= 70000
123    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	salary
Jane	Smith	80000.00
Jim	Taylor	78000.00
Amy	Adams	70000.00

다중 행 서브쿼리 (Multi-row Subquery)

- 모든 부서의 평균 급여보다 높은 급여를 받는 직원들을 찾으세요.**(ALL)**

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary > ALL (
    SELECT AVG(salary)
    FROM employees
    GROUP BY department_id
);
```

```
125 •  SELECT first_name, last_name, salary
126    FROM employees
127    WHERE salary > ALL (
128        SELECT AVG(salary)
129        FROM employees
130        GROUP BY department_id
131    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	salary
Jane	Smith	80000.00

다중 열 서브쿼리 (Multi-column Subquery)

- 여러 열을 반환하는 서브쿼리로, **WHERE** 절에서 여러 열을 비교할 때 사용

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE (department_id, job_id) IN (
    SELECT department_id, job_id
    FROM employees
    WHERE employee_id = 101
);
```

```
82 • SELECT employee_id, first_name, last_name
83   FROM employees
84   WHERE (department_id, job_id) IN (
85       SELECT department_id, job_id
86       FROM employees
87       WHERE employee_id = 101
88   );
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	first_name	last_name
101	John	Doe
103	Sam	Brown

```
90 •  SELECT department_id, job_id
91      FROM employees
92      WHERE employee_id = 101;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

department_id	job_id
1	Salesman

상관 서브쿼리 (Correlated Subquery)

- 메인 쿼리의 각 행에 대해 한 번씩 실행되며, 메인 쿼리의 데이터를 참조

```
SELECT e1.employee_id, e1.first_name, e1.last_name
FROM employees e1
WHERE salary > (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e1.department_id = e2.department_id
);
```

메인쿼리에 상관있는 데이터를 찾아 그것을 기준으로 메인쿼리에서 찾음

```
94 •  SELECT e1.employee_id, e1.first_name, e1.last_name
95    FROM employees e1
96    WHERE salary > (
97        SELECT AVG(salary)
98        FROM employees e2
99        WHERE e1.department_id = e2.department_id
100   );
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	first_name	last_name
102	Jane	Smith
103	Sam	Brown

스칼라 서브쿼리 (Scalar Subquery)

- 하나의 값을 반환하는 서브쿼리로, **SELECT 절, WHERE 절, HAVING 절**에서 사용

```
SELECT employee_id, first_name, last_name,
        (SELECT department_name FROM departments
        WHERE department_id = employees.department_id) AS
        department_name
        FROM employees;
```

```
102 •  SELECT employee_id, first_name, last_name,
103            (SELECT department_name FROM departments
104    WHERE department_id = employees.department_id) AS department_name
105    FROM employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	first_name	last_name	department_name
101	John	Doe	Sales
102	Jane	Smith	Engineering
103	Sam	Brown	Sales
104	Sue	Wilson	HR
105	Jim	Taylor	Engineering
106	Amy	Adams	Finance

EXISTS 서브쿼리

- 서브쿼리가 레코드를 반환하는지 여부를 확인

```
SELECT first_name, last_name
FROM employees
WHERE EXISTS (
    SELECT 1
    FROM departments
    WHERE departments.department_id = employees.department_id
    AND location_id = 1700
);
```

조건이 맞으면 1을
출력O, 아니면 출력X
→ select *는 불필요한
컬럼 조회를 하게 됨.

```
108 • SELECT first_name, last_name
109   FROM employees
110   WHERE EXISTS (
111       SELECT 1
112       FROM departments
113       WHERE departments.department_id = employees.department_id
114       AND location_id = 1700
115   );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
John	Doe
Sam	Brown
Sue	Wilson

EXISTS와 SELECT 1의 작동 방식

- **SELECT 1의 역할**
  - EXISTS 서브쿼리 내에서 SELECT 1은 레코드가 존재하는지 여부를 체크
  - 서브쿼리가 한 개 이상의 레코드를 반환하면 EXISTS 절은 참(True)을 반환하고, 그렇지 않으면 거짓(False)을 반환
- **왜 SELECT 1을 사용하는가**
  - SELECT 1은 효율적
  - 어떤 열을 반환하는지가 중요하지 않기 때문에, 단순히 1을 선택하여 존재 여부만을 확인
  - 실제로 SELECT *를 사용해도 동일하게 작동하지만, SELECT 1은 불필요한 데이터를 조회하지 않으므로 더 간결하고 효율적

서브 쿼리 확인 문제

- 'HR' 부서의 ID를 조회하여 해당 부서에 속한 직원들의 이름과 급여를 출력하세요.(단일 행

```
SELECT first_name, last_name, salary
FROM employees
WHERE department_id = (
    SELECT department_id
    FROM departments
    WHERE department_name = 'HR'
);
```

```
135 •  SELECT first_name, last_name, salary
136    FROM employees
137    WHERE department_id = (
138        SELECT department_id
139        FROM departments
140        WHERE department_name = 'HR'
141    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	salary
Sue	Wilson	55000.00

서브 쿼리 확인 문제

- location_id가 1800인 부서에 속한 직원들의 이름과 부서 ID를 출력하세요.(다중 행

```
SELECT first_name, last_name, department_id
FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM departments
    WHERE location_id = 1800
);
```

```
143 •  SELECT first_name, last_name, department_id
144    FROM employees
145    WHERE department_id IN (
146        SELECT department_id
147        FROM departments
148        WHERE location_id = 1800
149    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	department_id
Jane	Smith	2
Jim	Taylor	2

서브 쿼리 확인 문제

- 특정 직원(employee_id=104)의 department_id와 job_id가 같은 다른 직원들의 ID와 이름을 조회하세요.(다중 열 서브쿼리)

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE (department_id, job_id) IN (
    SELECT department_id, job_id
    FROM employees
    WHERE employee_id = 104
);
```

```
151 •  SELECT employee_id, first_name, last_name
152    FROM employees
153    WHERE (department_id, job_id) IN (
154        SELECT department_id, job_id
155        FROM employees
156        WHERE employee_id = 104
157    );
158
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	first_name	last_name
104	Sue	Wilson

서브 쿼리 확인 문제

- 동일한 부서에서 평균 급여보다 낮은 급여를 받는 직원들의 ID와 이름을 조회하세요.(상관 서브쿼리)

```
SELECT e1.employee_id, e1.first_name, e1.last_name
FROM employees e1
WHERE salary < (
    SELECT AVG(salary)
    FROM employees e2
    WHERE e1.department_id
                = e2.department_id
);
```

```
160 •  SELECT e1.employee_id, e1.first_name, e1.last_name
161    FROM employees e1
162    WHERE salary < (
163        SELECT AVG(salary)
164        FROM employees e2
165        WHERE e1.department_id = e2.department_id
166    );
167
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	first_name	last_name
101	John	Doe
105	Jim	Taylor

서브 쿼리 확인 문제

- 각 직원의 이름과 그들이 속한 부서의 location_id를 조회하세요.(스칼라 서브쿼리)

```
SELECT employee_id, first_name, last_name,
    (SELECT location_id FROM departments
        WHERE department_id = employees.department_id)
        AS location_id
FROM employees;
```

```
168 • SELECT employee_id, first_name, last_name,
169       (SELECT location_id FROM departments
170           WHERE department_id = employees.department_id)
171           AS location_id
172   FROM employees;
173
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	first_name	last_name	location_id
101	John	Doe	1700
102	Jane	Smith	1800
103	Sam	Brown	1700
104	Sue	Wilson	1700
105	Jim	Taylor	1800
106	Amy	Adams	1900

서브 쿼리 확인 문제

- location_id가 1900인 부서에 속한 직원들의 이름을 출력하세요.(EXISTS 서브쿼리)

```
SELECT first_name, last_name
FROM employees
WHERE EXISTS (
    SELECT 1
    FROM departments
    WHERE departments.department_id
                = employees.department_id
    AND location_id = 1900
);
```

```
174 •  SELECT first_name, last_name
175    FROM employees
176    WHERE EXISTS (
177        SELECT 1
178        FROM departments
179        WHERE departments.department_id = employees.department_id
180        AND location_id = 1900
181    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
Amy	Adams

서브쿼리 (insert)

- 서브쿼리를 사용하여 다른 테이블에서 데이터를 선택하여 새로운 테이블에 삽입하는 예제

```
-- 테이블 구조
-- employees (employee_id, first_name, last_name, department_id)
-- new_employees (employee_id, first_name, last_name, department_id)

INSERT INTO new_employees (employee_id, first_name, last_name, department_id)
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id = 10;
```

서브쿼리 (delete)

- 서브쿼리를 사용하여 특정 조건에 맞는 레코드를 삭제하는 예제

```
-- 테이블 구조
-- employees (employee_id, first_name, last_name, department_id)
-- departments (department_id, location_id)

DELETE FROM employees
WHERE department_id IN (
    SELECT department_id
    FROM departments
    WHERE location_id = 1700
);
```

서브쿼리 (update)

- 서브쿼리를 사용하여 다른 테이블에서 값을 가져와 업데이트하는 예제

```
-- 테이블 구조
-- employees (employee_id, first_name, last_name, department_id, salary)
-- departments (department_id, manager_id)

UPDATE employees
SET salary = salary * 1.1
WHERE department_id IN (
    SELECT department_id
    FROM departments
    WHERE manager_id = 200
);
```

핵심 정리

- **주요함수**
  - 문자, 숫자, 날짜
  - 형식변환
- **그룹함수**
  - sum(), avg(), min(), max(), count()
  - group by, having
- **집합**
- **조인/UNION**
- **subquery/IN**