MySQL

Oracle

> 오라클이란 미국의 오라클 회사에서 제작한 세계 점유율 1위 데이터베이스 관리 시스템이며 현재 유닉스 체제에서 가장 많이 사용되는 DBMS이다.

MySQL

> MySQL은 전세계적으로 가장 널리 사용되고 있는 오픈 소스 데이터베이스이며, MySQL AB사가 개발하여 배포 및 판매하고 있는 데이터베이스 관리툴이다.

Oracle | MySQL 차이점

구조적 차이

Oracle: DB 서버가 통합 하나의 스토리지를 공유하는 방식

MySQL: DB 서버마다 독립적인 스토리지를 할당하는 방식

조인 방식의 차이

Oracle: 중첩 루프 조인, 해시 조인, 소트 머지 조인 방식을 제공

MySQL: 중첩 루프 조인 방식을 제공

확장성의 차이

Oracle: 별도의 DBMS을 설치해 사용할 수 없음

MySQL: 별도의 DBMS을 설치해 사용할 수 있음

메모리 사용율의 차이

Oracle: 메모리 사용율이 커서 최소 수백MB 이상이 되어야 설치 가능

MySQL: 메모리 사용율이 낮아서 1MB 환경에서도 설치가 가능

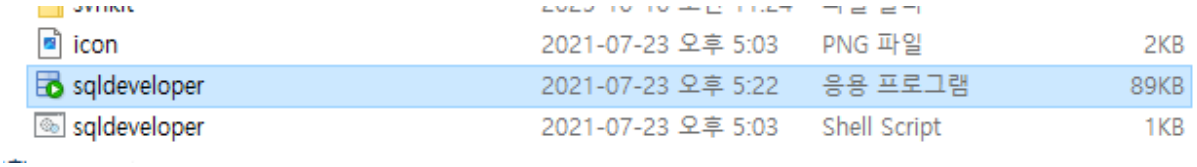
👍 **한줄요약** **(대기업 - ORALCE 사용 , 중소기업 - My-Sql 사용)**

- Oracle은 대기업용 유료 DBMS로 대규모 데이터 처리에 강점
- MySQL은 중소기업용 오픈소스로 가벼운 웹 애플리케이션에 적합

📌 OracleDeveloper - “SQL(스쿼데브)”

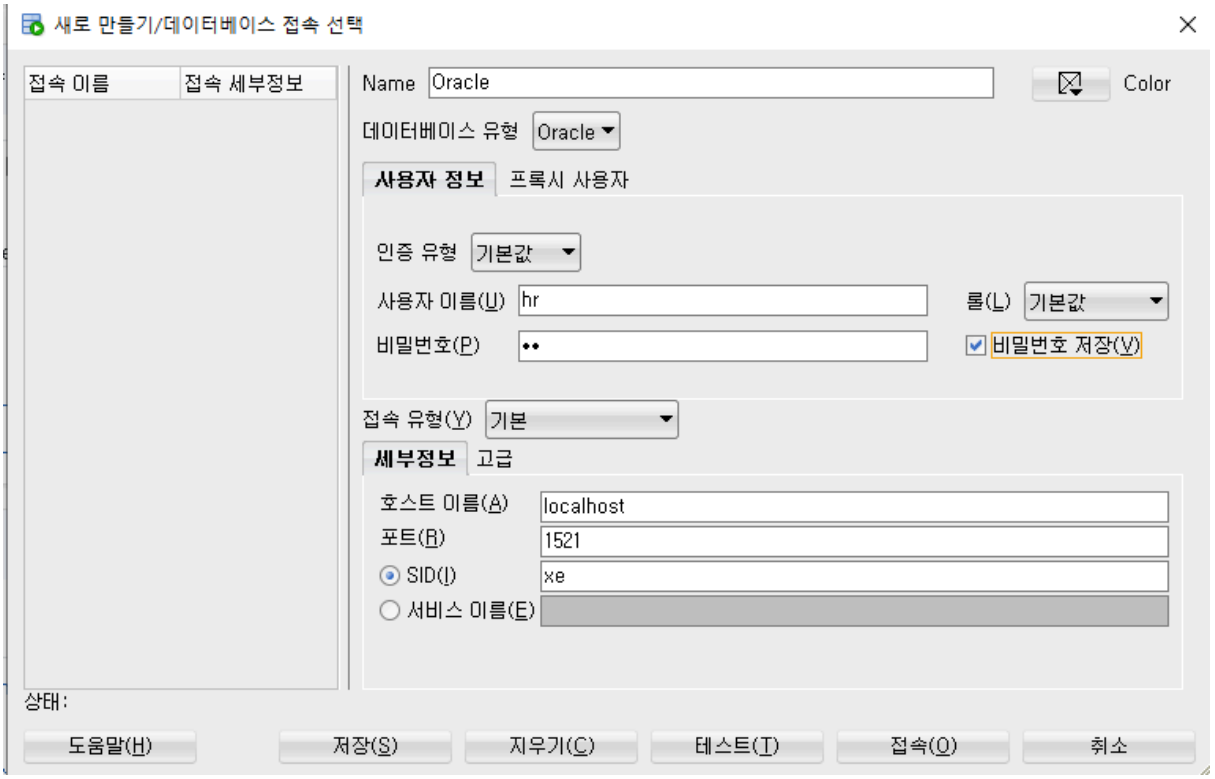
SQL - Developer 툴 다운

https://naver.me/5jBeC7wV - 1111

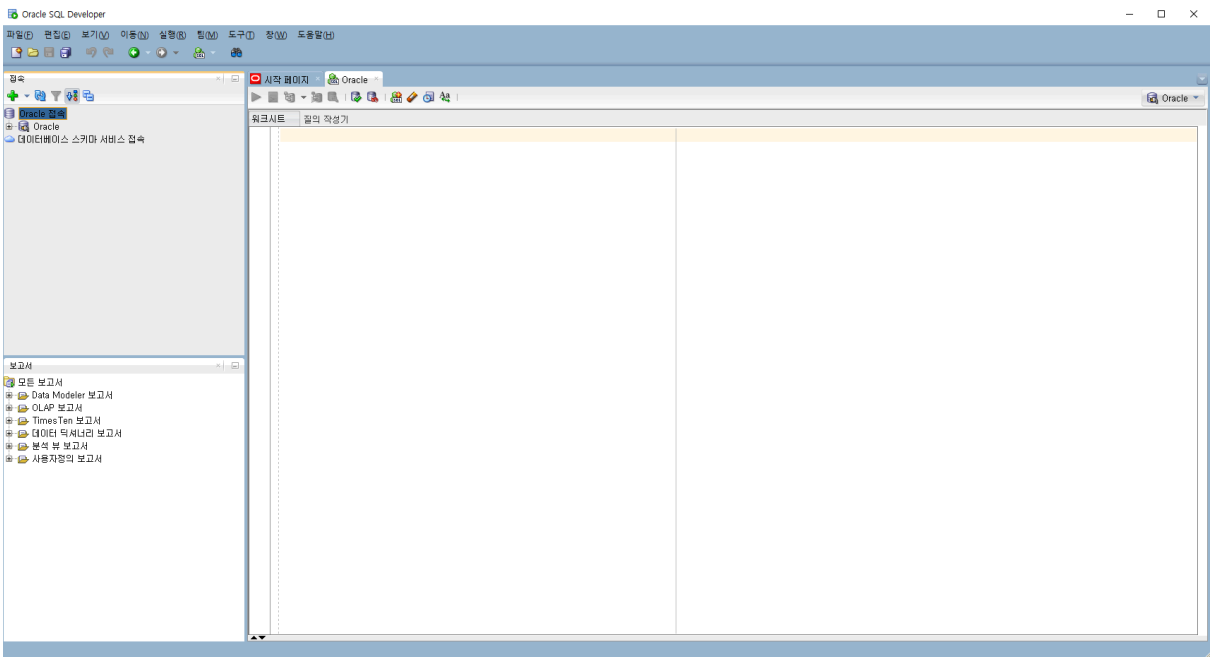


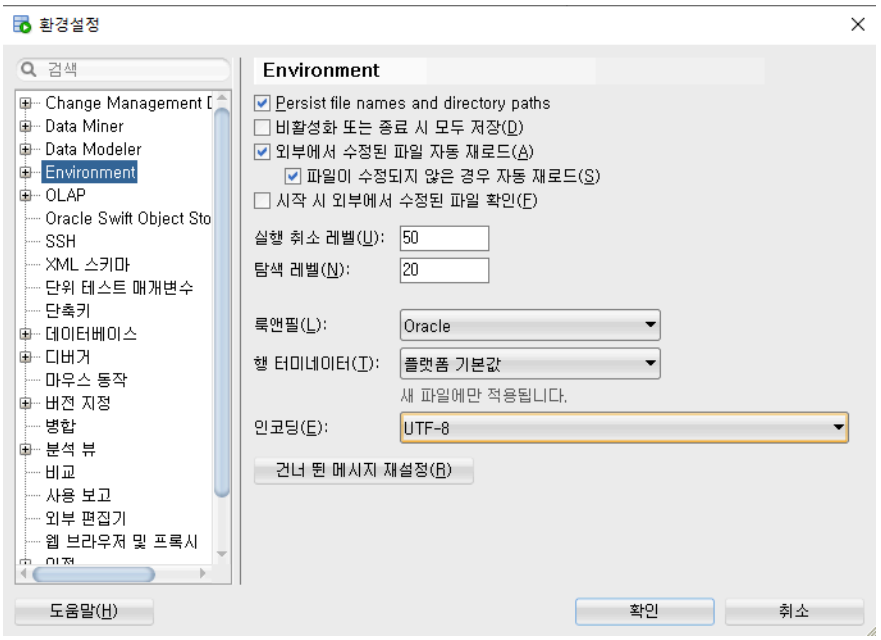
실행

Oracle - HR



접속 누르고 만들때 (반드시 비밀번호 저장 Check !!! : 잃어버리면 엄청 피곤해짐)





📌 데이터 베이스 (MySQL)

Jan 2025	Jan 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.28%	+9.32%
2	3	↑	C++	10.29%	+0.33%
3	4	↑	Java	10.15%	+2.28%
4	2	↓	C	8.86%	-2.59%
5	5		C#	4.45%	-2.71%
6	6		JavaScript	4.20%	+1.43%
7	11	⇈	Go	2.61%	+1.24%
8	9	↑	SQL	2.41%	+0.95%
9	8	↓	Visual Basic	2.37%	+0.77%
10	12	↑	Fortran	2.04%	+0.94%

2025년 프로그래밍언어 순위 8위에 MySQL이 있다.

SQL(Structured Query Language)은 '**구조화된 질의 언어'로**

관계형 데이터베이스 관리 시스템(RDBMS)에서 데이터를 관리하고

조작하기 위해 사용하는 표준 프로그래밍 언어로 손 꼽힌다.

✨ My-SQL 테이블 구조와 용어

세로 방향의 항목 이름. no, name, addr가 컬럼명 이다.

가로 한 줄이 하나의 사람 기록. 1 / k / s 가 한 행(레코드, 튜플)이

```
no | name | addr   ← 컬럼명 (세로 항목)
1 |  k   |  s     ← 컬럼값들 =  컬럼값 합친것을 행(레코드셋 / 튜플)
```

MySQL의 RDB(관계형 데이터베이스)에서 ERD(Entity Relationship Diagram)는 개체- 테이블과 테이블 간의 관계를 시각적으로 표현한 것

✨데이터베이스 작동원리

- **Database**: 데이터를 구조적으로 저장해 두는 곳(창고).
- **DBMS**: DB를 만들고 읽고 바꾸고 지키는 소프트웨어(Oracle, MySQL 등).
- **RDBMS**: 테이블(행·열) + 키(PK/FK) + SQL로 다루는 DBMS.
- **스키마**: 테이블/컬럼/제약/인덱스 같은 **구조(설계서)**.
- **테이블**: 같은 형태의 행이 모여 있는 표.
- **열(컬럼)**: 데이터의 항목 이름/타입.
- **행(레코드/튜플)**: 한 개체(사람 1명, 주문 1건) 데이터 묶음.
- **도메인(domain)**: 컬럼이 가질 수 있는 **값의 범위/타입 규칙**(URL 아님!).
- **ERD**: 엔터티(표 후보)·속성(컬럼 후보)·관계(FK)를 그린 개념 설계도.

.

1. Entity - 업무에서 관리할 대상. 회원, 주문, 상품처럼 “무엇”을 뜻한다.
2. Attribute(속성) - 그 대상의 성질. 회원.이메일, 주문.금액처럼 컬럼 후보.
3. Relationship(관계) - 엔터티 간 연결. 1:N이 기본, M:N은 교차 테이블로 풀고 1:1은 선택·종속에 따라 결정.

ERD란?
엔터티·속성·관계를 그린 개념 설계도. DBMS와 언어(Java)와는 독립.

이걸 바탕으로 실제 **스키마(Schema)**를 만든다.

스키마(Schema)란?

일반 의미에선 “데이터 구조와 제약의 설계서”

엔터티·속성·관계까지 포괄하는 개념을 말할 수 있다

RDB란?
테이블(행·열)로 데이터를 저장하고 PK·FK로 관계와 무결성을 지키는 데이터베이스.

SQL로 조인·집계를 자유롭게 함.

ERD의 엔터티 → RDB의 테이블 → Java의 클래스

ERD의 속성 → RDB의 컬럼 → Java의 필드

ERD의 관계 → RDB의 외래키 → Java의 참조(단일)·컬렉션(다수)

pk = primary key : 주요 키

fk = foreign key : 참조 키

pk = Primary Key (주민등록번호 같은 개념)

테이블에서 각 행을 유일하게 식별하는 키. 중복 불가, NULL 불가.

한 테이블에 하나의 PK 제약, 단일 또는 복합 컬럼 가능. 값 변경은 최소화하는 것이 원칙이다.

fk = Foreign Key
다른 테이블의 PK 또는 UNIQUE 키를 참조해 관계와 정합성을 보장하는 키.

부모에 없는 값은 입력 불가. 부모 키 삭제·변경 시 규칙에 따름
옵션 예시 ON DELETE/UPDATE CASCADE, SET NULL, RESTRICT
정의 위치는 자식 테이블, 성능을 위해 FK 컬럼 인덱스 권장.

ERD 설계(데이터간의 관계) - 테이블 설계를 잘해야한다.

대리키(id) 사용, FK에 인덱스, 컬럼의 NULL 허용 여부를 명확히, 1:N은 항상 N쪽에 **FK**.

EX) “회원 1 — N 주문”을 ERD로 잡으면, 실제 스키마에선 회원(id PK), 주문(id PK, member_id FK) 두 테이블로 구현한다.

고객의 식별자는 고객 PK, 주문은 그 고객 PK를 FK로 참조, “주문에 담긴 상품들”은 교차 테이블(order_items)에서 주문 FK와 상품 FK 두 개로 표현한다.

~~아~! 이런 흐름이구나로 우선 생각하고 실습부터 차례대로 진행해보자~~

💡데이터베이스.sql (실습1)

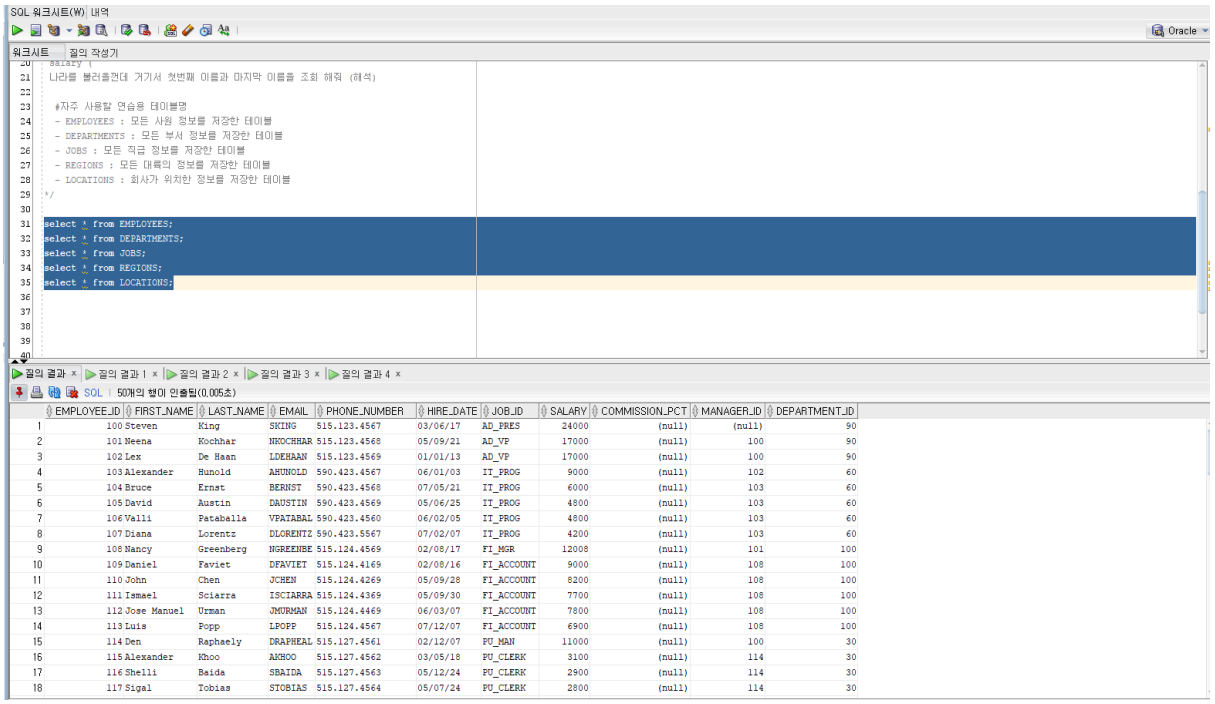
Ctrl + Enter : 커서 위치의 쿼리문 실행 (테이블 출력)

```
-- 데이터베이스.sql 전체 코드
select * from tab; -- Ctrl + Enter : 커서 위치의 쿼리문 실행 / (탭)
select * from employees; -- 연습용 고객정보 테이블 ( employees - 직원 )
select * from all_users; -- 연습용 회원정보 테이블 ( users - 회원 )
/*
    # 데이터베이스 (Database)
    - 데이터 창고
    - 데이터를 효율적으로 저장하기 위한 데이터 저장 전문 프로그램
    - 파일 시스템의 많은 문제점과 한계를 극복하기 위한 프로그램

    # 파일 시스템의 문제점
    - 데이터 불일치가 발생할 수 있음
    - 다수 사용자를 위한 동시 제공이 불가능 함
    - 중복 데이터를 필요 이상으로 많이 저장하게 됨
    - 파일 복구 기능이 없음

    # DBMS (Database Mangement System)
    - 데이터베이스를 보관하고, 관리하기 위한 프로그램
    - 데이터를 다루는 작업은 DMBS가 함
```

```
    # RDBMS (Relational(관계) Database Mangement System)
    - 데이터들 간의 관계를 이용하여 데이터의 중복을 최소화하는 방식의 DMBS
    - 질의문(Query, SQL)으로 데이터를 관리함
    - 쿼리는 국제 표준을 따름
    - 데이터를 표(table) 형태로 저장

    #테이블
    - 관계형 데이터베이스는 데이터를 표 형태로 저장
    - 필드 : 한 열에 저장되는 데이터들의 이름 = attribute = 속성
    - 레코드 : 한 행에 저장되는 하나의 개체에 대한 데이터들의 묶음 = tuple = *튜플

    # SQL (Struectured Query Language)
    - RDBMS에 명령을 내기리 위한 언어
*/

/*
  # 오라클 DB 이름 명명 규칙
  a. 문자로 시작해야함
  b. 1자부터 30자까지 가능
  c. A-z, a-z, 0-9, _, $ 포함 가능
  d. 동일한 사용자가 소유한 다른 DB의 이름과 중복되지 않아야 함
  e. Oracle의 예약어가 아니어야함
*/

/* 기본 명령어 */
-- 현재 접속한 계정을 확인하는 명령어
show user;
-- 현재 접속한 계정이 가지고 있는 모든 테이블을 확인하는 명령어
select * from tab;

-- 테이블의 모든 내용을 확인하는 명령어

select * from employees; -- ( 직원 정보 )
select * from countries; -- ( 국가 정보 )
select * from departments; -- ( 부서 정보 )
```

실행 결과

```sql
select * from tab; -- Ctrl + Enter : 커서 위치의 쿼리문 실행 / (탭)
select * from employees; -- 연습용 고객정보 테이블 ( employees - 직원 )
select * from all_users; -- 연습용 회원정보 테이블 ( users - 회원 )

/*
    # 데이터베이스 (Database)
    - 데이터 창고
    - 데이터를 효율적으로 저장하기 위한 데이터 저장 전문 프로그램
    - 파일 시스템의 많은 문제점과 한계를 극복하기 위한 프로그램

    # 파일 시스템의 문제점
    - 데이터 불일치가 발생할 수 있음
    - 다수 사용자를 위한 동시 제공이 불가능 함
    - 중복 데이터를 필요 이상으로 많이 저장하게 됨
    - 파일 복구 기능이 없음
*/
```



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	03/06/17	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	05/09/21	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	01/01/13	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	06/01/03	IT_PROG	9000	(null)	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	07/05/21	IT_PROG	6000	(null)	103	60
6	105	David	Austin	DAUSTIN	590.423.4569	05/06/25	IT_PROG	4800	(null)	103	60
7	106	Valli	Pataballa	VPATABAL	590.423.4560	06/02/05	IT_PROG	4800	(null)	103	60
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/07	IT_PROG	4200	(null)	103	60
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	02/08/17	FI_MGR	12008	(null)	101	100
10	109	Daniel	Faviet	DFAVIET	515.124.4169	02/08/16	FI_ACCOUNT	9000	(null)	108	100
11	110	John	Chen	JCHEN	515.124.4269	05/09/28	FI_ACCOUNT	8200	(null)	108	100
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	05/09/30	FI_ACCOUNT	7700	(null)	108	100
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	06/03/07	FI_ACCOUNT	7800	(null)	108	100
14	113	Luis	Popp	LPOPP	515.124.4567	07/12/07	FI_ACCOUNT	6900	(null)	108	100
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	02/12/07	PU_MAN	11000	(null)	100	30
16	115	Alexander	Khoo	AKHOO	515.127.4562	03/05/18	PU_CLERK	3100	(null)	114	30
17	116	Shelli	Baida	SBAIDA	515.127.4563	05/12/24	PU_CLERK	2900	(null)	114	30
18	117	Sigal	Tobias	STOBIAS	515.127.4564	05/07/24	PU_CLERK	2800	(null)	114	30

SQL | 50개의 행이 인출됨(0.019초)

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	03/06/17	AD_PRES	
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	05/09/21	AD_VP	
3	102	Lex	De Haan	LDEHAAN	515.123.4569	01/01/13	AD_VP	
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	06/01/03	IT_PROG	
5	104	Bruce	Ernst	BERNST	590.423.4568	07/05/21	IT_PROG	
6	105	David	Austin	DAUSTIN	590.423.4569	05/06/25	IT_PROG	
7	106	Valli	Pataballa	VPATABAL	590.423.4560	06/02/05	IT_PROG	
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/07	IT_PROG	
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	02/08/17	FI_MGR	
10	109	Daniel	Faviet	DFAVIET	515.124.4169	02/08/16	FI_ACCOUNT	
11	110	John	Chen	JCHEN	515.124.4269	05/09/28	FI_ACCOUNT	
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	05/09/30	FI_ACCOUNT	
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	06/03/07	FI_ACCOUNT	
14	113	Luis	Popp	LPOPP	515.124.4567	07/12/07	FI_ACCOUNT	
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	02/12/07	PU_MAN	
16	115	Alexander	Khoo	AKHOO	515.127.4562	03/05/18	PU_CLERK	
17	116	Shelli	Baida	SBAIDA	515.127.4563	05/12/24	PU_CLERK	

💡SELECT.sql (실습2)

```
/*
  # select 컬럼명 from 테이블명;
    - 원하는 테이블을 조회하는 쿼리문
    - 컬럼명 자리에 *을 쓰는 것은 모든 컬럼을 의미
    - 컬럼명과 테이블명은 대소문자를 구분하지 않음
    - 쿼리문은 대소문자를 구분하지 않음
    - 단, 데이터는 대소문자를 구분함
*/

-- employees 테이블의 모든(*) 정보를 선택해서 조회 쿼리문 실행
select * from employees;

-- 조회하고 싶은 컬럼의 값만 가져오는 쿼리문
select first_name, last_name, hire_date, salary, job_id from employees;

/*
   # 자주 사용할 연습용 테스트 테이블명
     -  employees: 사원
     -  departments: 부서
     -  jobs: 직급
     -  regions: 대륙
     -  locations:위치
*/

-- 위 5개 테이블의 모든 정보 출력해보기 => 쿼리문
select * from employees;  -- 모든 사원 정보를 저장한 테이블
select * from departments; -- 모든 부서 정보를 저장한 테이블
select * from jobs; -- 모든 직급 정보를 저장한 테이블
select * from regions; --  각 대륙의 정보를 저장한 테이블
select * from locations; -- 회사가 위치한 정보를 저장한 테이블

-- 해당 테이블 컬럼의 정보 보기
DESC employees;
DESC departments;
DESC jobs;
DESC regions;
```

```
DESC locations;

/*
   # SQL 기본 데이터 타입

   # NUMBER(n), NUMBER(n,m)
     - 숫자 데이터만 저장할 수 있는 컬럼 타입
     - 숫자가 하나만 적혀 있으면 정수를 저장하는 컬럼임
     - 숫자가 2개 적혀 있으면 실수를 저장하는 컬럼임
       (예) NUMBER(8)   → 정수 8자리
            NUMBER(8,2) → 정수 6자리, 소수 2자리

   # CHAR(n)
     - 고정 길이 문자 데이터를 저장하는 컬럼 타입
     - 설정된 컬럼 크기보다 적은 양의 데이터가 들어오더라도 설정된 길이를 모두 차지
함
     - 데이터가 낭비될 수 있지만, 크기 계산이 없기 때문에 속도는 좀 더 빠름

   # VARCHAR2(n)
     - 가변 길이 문자 데이터를 저장하는 컬럼 타입
     - 저장하는 데이터의 크기에 때라 알맞은 공간을 사용함
     - 데이터를 저장할 때 저장공간이 절약되지만 크기 계산이 필요함

   # DATE
     - 날짜 및 시간을 저장하는 컬럼 타입
*/

-- 실습1. 모든 사원의 사번/이름(2개모두)/입사일/월급/부서번호 조회
select employee_id, first_name, last_name, hire_date, salary, department_id
from employees;

-- AS 통해 해당 컬럼 이름 변경 조회 가능
select
   employee_id AS 사번,
   first_name AS 이름,
   salary AS 급여,
   department_id AS 부서번호
from
```

```
    employees;


-- 산술 연산자 이용

-- 모든 사원의 마지막 이름, 월급만 나오도록 조회
select last_name, salary from employees;

-- 모든 사원의 마지막 이름, 월급 컬럼 별칭 "이사람의 연봉" 계산하여 조회
select last_name, salary*12 AS "이 사람의 연봉" from employees;

-- 모든 사원의 마지막 이름, 월급 컬럼 별칭 "20퍼센트 삭감" 계산하여 조회
select last_name, salary*0.8 AS "20퍼센트 삭감" from employees;

/*
    # NVL = null values
      - 계산에 사용되는 컬럼의 값에 null이 있는 경우,
      null을 대체할 값(=치환)을 지정하는 함수

    #commission_pct (수수료)

    #distinct = 중복되는 내용 한번씩만 출력
*/

select
    last_name,
    salary,
    commission_pct,
    job_id,
    salary * (1+NVL(commission_pct,0)) AS "수수료 적용 월급"
from
    employees;

-- 모든 사원들의 사번,이름,직책ID, 별칭 "보너스가 적용된 연봉"을 출력
select
    employee_id, -- 사원번호
    first_name, -- 이름
    job_id, -- 직책
```

```
    salary * (1+NVL(commission_pct,0))*12 AS "보너스 적용된 연봉" -- 연봉
from
    employees;



-- 모든 사원들의 직무 조회
select job_id from employees;

-- 직무별로 한개씩만 출력 = 중복값 제거
select distinct job_id from employees;

-- 사원들의 모든 부서ID 조회
select department_id from employees;

-- 부서ID 한번씩 조회(중복값 제거)
select distinct department_id from employees;


-- 계정의 모든 테이블명 출력
select * from TAB;
select * from employees;

/* # AND, OR, NOT */
select * from employees
 where hire_date >= DATE '2006-05-01'
 and hire_date <  DATE '2008-09-01';
 -- 2006-05-01 이상, 2008-08-31까지 포함


select * from employees
 where job_id ='IT_PROG'
 OR job_id = 'SH_CLERK';  --

select * from employees
 where job_id ='IT_PROG'  -- IT_PROG 만

select * from employees
```

```
 where NOT job_id ='IT_PROG'  -- IT_PROG 가 아닌


-- 실습 2000에서 3000 사이의 월급을 받는 직원들 정보 조회
select * from employees
where salary > 2000 AND salary < 3000;


-- 실습 30번, 60번, 90번 부서에 소속된 직원들의 이름/ 직무 / 전화번호 / 부서번호 정보 조회
select  first_name, job_id,  phone_number
from employees
where department_id = 30 or  department_id = 60 or department_id = 90;


/* #컬럼명 BETTWEEN A AND B = 해당 컬럼의 값이 A이상 B이하인 경우 True */
select * from employees where salary BETWEEN 2000 AND 3000;

/* #null = (무한대) 크기 비교가 불가능하여 비교연산자 사용 불가 */

-- commission_pct가 0.2 미만인 모든 직원 정보 출력
select * from employees where commission_pct < 0.2;
select * from employees where commission_pct = null; -- 조회안됨
select * from employees where commission_pct is null; -- is null 사용

-- 사원번호가 100번 모든 정보 조회
select * from employees where employee_id = 100;

/* NOT 연산자의 위치는 비교적 자유롭다 = 모두 같은 결과*/
select * from employees where commission_pct is not null;
select * from employees where not commission_pct is null;
select * from employees where not salary  BETWEEN 2000 AND 3000 ;
select * from employees where salary  not BETWEEN 2000 AND 3000 ;

/*
  #LIKE
  - 데이터의 일부분으로 원하는 애용을 검색할 수 있음
  - 문자 타입과 날짜 타입에 사용할 수 있음
```

```
  - 와일드 카드
   %(퍼센트) = 길이 제한 없이 아무 문자가 와도 상관없는 와일드 카드 = 포함된 모든 문자 조회
   _(언더바) = 하나의 문자가 반드시 와야하는 와일드카드 = 하나의 문자만 대응
*/

-- 첫번째 이름중 대문자 D를 보여줘
select * from employees where first_name like 'D%'

-- 첫번째 이름중 두번째 이름의 소문자 "e"가 들어간 사람을 찾아줘
select * from employees where first_name like '_e%'

-- 이름안에 t들어간 모든 직원 검색
select * from employees where first_name like '%t%';

-- 이름에서 처음 나오는 a의 직원의 이름중 뒤에 3글자가 더 나오도록
select * from employees where first_name like '%a___';
-- a___ 포함 뒤 3글자

-- 이름에 e가 두개 이상 포함된 직원
select * from employees where first_name like '%e%e';

-- 고용일이 5월인 직원
select * from employees where hire_date like '%/05/%';

-- 고용일의 마지막 날짜가 5일인 직원
select * from employees where hire_date like '%/_5';
```

 # NVL(column,value) = 컬럼의 값에 null이 있는 경우, null을 대체할 값(=치환)을 지정하는 함수

 # LIKE 연산자

- **%: 0개 이상의 모든 문자를 대체 (글자 개수를 지정하지 않는 와일드 카드)**
- **_: 1개 이상의 모든 문자를 대체 (_ 개수 만큼 글자 수가 지정되는 와일드 카드)**

💡WHERE.sql (실습3)

```
/*
 #select 컬럼명 from 테이블명 where 조건절;
  - select문에 where절을 추가하여 해당 조건을 만족하는 행만 조회 가능
  - 오라클의 비교 연산자들을 활용

  # 비교 연산자
  - = : 같으면 TURE
  - < , > , <= , >= : 비교
  - != , <> , ^= : 다르면 TURE
*/

-- 모든 employees 테이블 조회
select * from employees

-- 직원 테이블에서 월급이 10000이상인 직원의 이름 모두 직무 급여를 조회
select first_name, last_name, job_id, salary
from employees
where salary >= 10000;


-- 모든 직원중에 첫번째 이름이 John인 사람의 모든 정보 출력
select *
from employees
where first_name like 'John%';

-- 모든 직원중에 첫번째 이름이 Vance인 사람의 부서정보, 급여, 마지막이름 출력
select department_id, salary, last_name
from employees
where first_name = 'Vance';

 /* #문자타입 비교 */

 -- 첫번째 이름이 Steven 이라는 직원의 모든 정보 조회
select * from employees
where first_name = 'Steven';

select  first_name
from employees
```

```
where first_name > 'P' ; -- p 부터~
order by first_name;

select *
from employees
where first_name > 'V'
order by first_name;


/* # 날짜 타입 비교 */

select * from employees
where hire_date < '06/01/01'
order by hire_date desc;

--모든 직원들의 월급중 2000에서 3000사이 직원 조회
select * from employees
where salary between 2000 and 3000;

-- 실습 30번, 60번, 90번 부서에 소속된 직원들의 이름/ 직무 / 전화번호 / 부서번호 정보 조회

select first_name, job_id ,  phone_number , department_id
from employees
where department_id = 30 or  department_id = 60 or department_id = 90;

select first_name, job_id , phone_number, department_id
from employees
where department_id 30 or department_id 60 or department_id 90


-- 모든 직원중  급여가 2000에서 3000 사이인 직원
select * from employees
where salary between 2000 and 3000;
```

논리 실행 순서는 from → where → select → order by

💡테이블생성.sql (실습4)

✨생성전 이론 : SQL 언어 크게 3가지 DDL, DML, DCL 정의

SQL 문장 분류

- **DML**: `select/insert/update/delete/merge` (데이터 다룸).
- **DDL**: `create/alter/drop/truncate` (구조 다룸).
- **DCL**: `grant/revoke` (권한).
- **TCL**: `commit/rollback/savepoint` (트랜잭션 제어).
-

DDL(데이터 정의 언어) - 데이터 구조(스키마·객체)를 생성·변경·삭제하는 명령.

DML(데이터 조작 언어) - 테이블의 행 데이터를 삽입·수정·삭제(일부 교재는 조회 포함)하는 명령.

DCL(데이터 제어 언어) - 사용자·역할 권한을 부여·회수하여 접근을 제어하는 명령.

#DML (Data Manipulation Language) = 데이터 조작어 (DML *암기 할것)

- SELECT : 데이터 조회
- INSERT : 데이터 추가
- UPDATE : 데이터 수정
- DELETE : 데이터 삭제

#DDL (Date Definition Language) = 데이터 정의어 (DDL *암기 할것)

- 테이블, 시퀀스, 뷰 ...등 DB에서 사용하는 DB 오브젝트 구조를 생성할 때 사용하는 명령어
- CREATE : 오브젝트 생성
- DROP : 오브젝트 삭제
- ALTER : 오브젝트 수정
- TRUNCATE : 오브젝트 완전삭제

#DCL (Data Control Language) = 데이터 제어 명령어 (DCL *암기 할것)

- GRANT : 권한 부여
- REVOKE : 권한 회수

char는 고정형이며, varchar는 가변형 길이를 말한다.

```
/*
   # 테이블 생성
     - CREATE TABLE 테이블명(컬럼명1 컬럼타입1 제약사항1,컬럼명2 컬럼타입2 제
약사항2,...);
   # 테이블 삭제
     - DROP TABLE 테이블명;
*/

-- 테이블 생성
create table fruits(
   name    VARCHAR2(20),
   qty     NUMBER(5),
   price   NUMBER(5)
);

select * from fruits;
-- 테이블 구조보기 : 명령어 desc
desc fruits;

-- 더미 데이터 삽입 : 명령어 insert into
insert into fruits VALUES('귤',40,1000);
insert into fruits values('사과',10,3500);
insert into fruits values('바나나',30,1800);

select * from fruits;

/* 데이터 수정 */

-- '귤' 이름을 '오렌지'로 업데이트
update fruits set name='오렌지' where name='귤';
```

```
-- 사과의 수량을 33개로 수정
update fruits set qty=33 where name='사과';

-- 데이터 삭제
DELETE FROM fruits where name='바나나';

-- 데이터 추가 망고 , 8, 2000
insert into fruits values('망고',8,2000);

-- 데이터 수정 갯수 12
update fruits set qty=12 where name='망고';

-- 데이터 삭제 망고
delete from fruits where name='망고';

commit; -- 영속성 (저장)
```

```sql
-- 테이블 생성(과일)
create table fruits(
  Name varchar2(20),    -- 과일명(문자). 20자까지
  qty  number(5),       -- 수량(정수). 최대 5자리
  price number(5)       -- 가격(정수). 최대 5자리
);

select * from fruits; -- 모든 행/모든 컬럼 조회
desc fruits; -- 테이블 구조 확인

insert into fruits values ('귤',40,1000);
insert into fruits values ('사과',30,50000);
insert into fruits values ('바나나',40,10000);
-- 테이블에 삽입 : 행 추가: NAME='귤', QTY=40, PRICE=1000.


COMMIT;
-- commit 해야 영구 저장됨.
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 3(0.002초)

	NAME	QTY	PRICE
1	귤	40	1000
2	사과	30	50000
3	바나나	40	10000

DB는 COMMIT; (브레이크 포인트 생성)으로 영속성 처리 꼭 해줘야한다 .

1.생성 (insert into 개체 values (배열,값))

```
-- 삽입 후르츠 테이블 vlaue 값들은 ( name , qty , price)
insert into fruits values('귤',40,1000);
```

```
insert into fruits values('사과',10,3500);
insert into fruits values('바나나',30,1800);
```

2.업데이트 (update: 개체 set 변경대상 = '값' where =대체대상 '값')

```
update fruits SET Name = '망고'  WHERE Name = '귤'; -- 과일 테이블 : 귤 > 망고로 변경

update fruits SET qty = 33 WHERE name = '사과'; -- 사과의 수량을 33개로 수정
```

삭제 (delete from 개체 where 변경대상 = '값')

```
delete from fruits where name = '바나나';
```

EX) 간단한 예제

```
insert into fruits values ('망고', 8,2000); -- 데이터 추가 망고 , 8, 2000
update fruits set qty = 12 where name = '망고'; -- 데이터 수정 갯수 12
delete from fruits where name = '망고'; -- 데이터 삭제 망고
```

코드 흐름 (한줄 요약)

- **UPDATE (해당 테이블 / 개체) SET (변경 대상) ='값' WHERE (바꿀 대상) = '값'**
- **ROLLBACK; 명령어 (이전 작업으로 돌리기)**

💡집합.sql (실습5)

- 기본값(오름차순) 생략:

```
select *
from employees
order by first_name; -- 생략
```

- 반대는 `desc`(내림차순)

```
select *
from employees
order by salary desc;
```

select department_id, salary, last_name = 보여줄 열만 고른다.

from employees = 데이터 소스 설정, “전체 직원” 범위를 잡는다.

where first_name = 'Vance' = 행 필터, 그 범위에서 조건에 맞는 행만 남긴다.

order by = 정렬 대상 / asc = 오름차순 (기본) desc = 내림차순

```
/*
   # UNION, INSERSEC, MINUS
*/

select * from employees;
-- 직원 테이블 이름순으로 기본정렬 = A부터 ~~
select * from employees order by first_name;


-- 직원테이블에서 직원의 이름이 J로 시작하고, s 또는 n으로 끝나는 직원들의 모든 정보를 조회 = AND , OR , LIKE
select *
from employees
where first_name LIKE 'J%'
AND (first_name LIKE '%s' OR first_name LIKE '%n');

-- 직원테이블에서 직원이름, 부서번호, 부서번호를 별칭으로 department_name으로 나오도록 조회 | 조건 = 부서번호에 10,20,30포함
select
   first_name,
   department_id,
   department_id AS 부서번호
from
   employees
where
```

```
   department_id IN(10,20,30);

/* UNION : 합집합 */
select * from employees where first_name LIKE 'J%n'
UNION
select * from employees where first_name LIKE 'J%s';

/* UNION ALL : 합집합 (중복제거안함) */
select * from employees where department_id = 30
UNION ALL
select * from employees where department_id between 10 and 30;

/* INTERSEC : 교집합 */
select * from employees where department_id = 30
INTERSECT
select * from employees where department_id BETWEEN 10 AND 30;

/* MINUS : 차집합 */
select * from employees where department_id between 10 and 30
MINUS
select * from employees where department_id = 30;
```

💡그룹함수.sql (실습6)

```
/*
   # 그룹함수
     - 테이블의 행들을 특정 컬럼 기준으로 그룹화하여 계산하는 함수들
     - 특정 그룹의 총합, 갯수, 평균 등을 구하는 함수들
     - 그룹의 기준이 되는 컬럼은 GROUP BY절을 통해 선택 (개념인지)
     - 그룹 함수의 결과는 일반 컬럼과 함께 출력될 수 있음
     ex ) group by department_id; --  그룹핑 (부서별)
*/

/*
 SUM(column)  : 합계
 AVG(column)  : 평균
```

```
 MAX(column)  : 최댓값
 MIN(column)  : 최솟값
 COUNT(column): 개수
*/

select * from employees;

-- 모든 직원 급여의 총 합계 조회
select SUM(salary) from employees;


-- 모든 직원의 그룹핑하여 급여의 총 합계 조회
select  job_id, SUM(salary)
from employees
group by job_id;


-- 각 직무별 평균임금을 직무 기준으로 조회 = 별칭 평균임균
select job_id,  AVG(salary) AS "평균임금"
from  employees
group by job_id;


-- 커미션 갯수 조회 | 조회기준 = 부서번호
select  department_id,COUNT(commission_pct) AS "총갯수"
from employees
group by department_id;


-- 부서번호 80번이 어떤 직무를 하는지 조회
select job_id from employees where department_id = 80;

-- 부서번호 80번이 어떤 직무를 하는지 조회(결과 중복제거)
select DISTINCT job_id from employees where department_id = 80;


-- 실습. 각 부서별로 가장 최근에 입사한 사원의 날짜와 가장 오래전 입사한 사원의 날
짜를 조회 = 부서번호로 그룹 조회
```

```
select
    department_id,
    MAX(hire_date) as "최근 입사",
    MIN(hire_date) as "초기 입사"
from
    employees
group by
    department_id;
-- NULL 처리 = 위의 결과 그대로 나오면서 NULL인 부서명을 9999 나아도록 작성
select
    NVL(department_id,'99999'),
    max(hire_date) as "최근입사",
    min(hire_date) as "초기입사"
from
    employees
group by
    department_id;
-- 실습. 각 직무별 평균 연봉 구하기 | 직무별로 조회
select
    job_id,
    AVG(salary * (1+NVL(commission_pct,0))) AS "평균연봉"
from
    employees
group by
    job_id;

/* # HAVING절 = 그룹함수 결과에 대한 조건을 주고 싶을때 사용 */
SELECT
    job_id,
    AVG(salary) as "평균급여"
FROM
    employees
GROUP BY
    job_id
HAVING
    AVG(salary) >= 10000;
-- 실습. 5명 이하로 구성된 부서번호를 조회 | 그 부서의 최고 급여액 , 부서번호 기준으로 조회
```

```
select
    department_id,
    max(salary) as "최고급여액"
from
    employees
group by
    department_id
having
    COUNT(department_id) >= 5;

/* WHERE절과 GROPU BY 함께 사용 = WHERE절의 조건을 모든 행에 적용한 결과를 그룹화 */
select
    department_id,
    MIN(salary) AS "최저급여"
from
    employees
where
    salary >= 5000
group by
    department_id;
```

💡정렬.sql (실습7)

```
/*
    # ORDER BY
      - 원하는 컬럼 기준으로 정렬하여 조회할 수 있음
      - ORDER BY 컬럼명 [ASC|DESC]
      - ASC : 오름 차순, Ascending(기본값) = (예)1,2,3,4,5
      - DESC : 내림 차순, Decending(기본값) = (예)5,4,3,2,1
      - NULL : 오름차순으로 정렬하면 가장 나중에 출력되고, 내림차순으로 정렬하면 가장 먼저 출력됨
*/

select * from employees;
```

```
-- 부서번호로 오름차순 정렬 | 모든 직원 조회
select * from employees order by department_id;
-- 부서번호로 내림차순 정렬 | 모든 직원 조회
select * from employees order by department_id desc;

-- NULL 정렬
select * from employees order by commission_pct;
select * from employees order by commission_pct desc; -- 가장 위로 정렬

-- 각각 컬럼에 정렬
select * from employees order by job_id asc, hire_date desc;
--먼저 job_id 오름차순으로 정렬하고, 같은 job_id 안에서는 hire_date 내림차순(최근 입사일이 위)으로 정렬
```

💡데이터 추가, 수정, 삭제 .sql (실습8)

```
/*
   # 테이블 데이터 생성
   - INSERT INTO 테이블명 (컬럼명,...) VALUES (컬럼값,...);
   - 테이블명에 컬럼명을 생략하면 모든 컬럼을 순서대로 넣어줘야함
*/

select * from employees;
select * from departments;
select * from fruits;

-- 컬럼값 이름, 수량 , 가격순으로 삽
insert into fruits (name, qty, price) values ('포도', 10, 3500);
delete from fruits where name = '포도';

-- 변경할 일부 컬럼만 삽입
insert into fruits (name, price) values ('바나나', 4000);
insert into fruits (price) values (8000);
--컬럼명 생략시 테이블 컬럼 순서대로 넣어야함

insert into fruits  values ('사과',20,2500); -- 일반적 삽입
```

```
commit;


select * from fruits;
-- 서브 쿼리를 이용해  insert 할수 있음  (복붙)

insert into fruits(select * from fruits); -- fruits 쿼리문 한번더 복 붙

create table fruits2 as (select * from fruits); -- fruits2로 fruits을복사

insert into fruits2 (select * from fruits);  -- fruits2에  fruits있는것들을 삽입

select * from fruits2;


/*
  #테이블 데이터 수정
  - UPDATE 테이블명 SET 컬럼=값 WHERE 조건;
  - 조건을 만족하는 모든행을 수정함;
  - 조건을 안쓰면 모든 행을 수정;
  - 하나의 행을 구분할 수 있는 컬럼이 조건으로 사용되는 경우가 많음 (예, 기본키= P
K : Primary Key);
*/

-- 모든 과일명을 사과로 변경
update  fruits set name = '사과';
rollback;
commit;

--  수량이 10개이하인 과일의 가격을 10으로 변경
update  fruits set price = 10 where qty <= 10;


/*
  #테이블 데이터 삭제
  - delete from 테이블명 where 조건
  - 조건을 만족하는 모든 행을 삭제
  - 조건을 안쓰는 모든 행을 삭제
```

```
*/

select * from fruits;
-- 과일명 포도 데이터 삭제

delete from fruits where name = '포도';

commit;
```

💡제약조건.sql (실습9)

```
/*
 #데이터 제약조건

 #데이터 무결성
 - 데이터가 결함이 없는 성질
 - 정확성, 일관성, 유효성이 유지되는 데이터를 말한다.

  # 개체 무결성(PK)
  - 테이블의 데이터 = 반드시 각 행을 구분할 수 있어야함
  - 테이터의 개체 무결성을 지키기 위한 제약조건 = PK 사용

  #참조 무결성 (FK)
  - 참조 관계에 있는 데이터는 항상 일관된 값을 가져야 함
  - 참조 무결성을 지키기 위한 제약 조건 = FK 사용

  #데이터 무결성 제약 조건
  - NOT NULL (NN) : 해당 도메인(domain)에 NULL을 허용하지 않음
    URL = 도메인명(Domain_Name) = table = 개체 라고 통합적으로 생각하면 된다.
(겹침없이 유니크해야함)

  - UNIQUE : 해당 도메인에 중복되는 값을 허용하지않음
  - PRIMARY KEY : 해당 도메인 테이블의 기본키로 사용 = UNIQUE + NOT NULL
  - FOREIGN KEY: 해당 컬럼을 외래키로 설정
  - CHECK : 원하는 조건을 지정, 도메인 무결성을 유지함

  # 도메인 무결성
```

```
  - 하나의 도메인을 구성하는 개체들은 항상 타입이 일정해야 함
  - 테이블 컬럼 타입 설정 및 CHECK 제약조건을 통해 유지 가능함

 데이터 사전 (Date Dictionary) - Dictionary(사전)
 - 시스템 카탈로그라고도 하며, 사용 가능한 데이터 베이스 및 테이블의 정보를 가지고
   있는 시스템 테이블
   DBA만 추가 , 수정, 삭제가 가능 => 사용자는 조회만 가능
*/

-- 제약조건을 볼 수있는 데이터 딕셔너리 뷰(접속사용자 권한) - 시스템 카탈로그 = e
x) hr 계정
select * from user_constraints; -- 사용자_제약조건

-- 제약조건을 볼 수있는 데이터 딕셔너리 뷰(모든 사용자 권한)
select * from all_constraints; -- 모든_제약조건

/*
 #constraint_type (제약조건 타입)
 - P : Primary key
 - F : Foreign key
 - C : Check or NOT NULL
 - U : Uinque
*/

/*
 #1. 테이블 생성과 동시에 제약 조건 추가
  - 컬럼명 컬럼타입 제약조건 (이름이 자동으로 정해짐)
  - 컬럼명 컬럼타입 CONSTRAINT 제약조건명 : 제약조건
*/

drop table fruits3; -- 테이블 삭제

create table fruits3(
    name varchar2(20) NOT NULL,
    price number(5)  NOT NULL
);

select * from fruits3;
```

```
desc fruits3;

insert into fruits3 (name,price) values('grape',100);

-- 제약사항조건 조회시 테이블명 '반드시 대문자로'
select * from user_constraints where table_name = 'FRUITS3' ; -- 대문자

create table fruits4 (
  name  varchar2(20)
        constraint fruits4_name_uk UNIQUE
        constraint fruits4_name_nn NOT NULL,
  price number(5)
        constraint fruits4_price_nn NOT NULL
);

select * from fruits4;
insert into fruits4 (name,price) values ('grape',100);
select * from user_constraints where table_name = 'FRUITS4';

commit;


/*
   PK: 줄마다 주민등록번호 같은 것(겹치면 X, 빈값 X)
   NOT NULL: 빈칸 금지
   UNIQUE: 서로 달라야 함
   순서: 만들고 → 넣고 → 확인 끝!
*/

-- 테이블 생성
create table fruits5 (
  fid   number constraint fruits5_pk primary key,             -- PK
  fname varchar2(50)ㅈ
        constraint fruits5_fname_nn not null                  -- NOT NULL
        constraint fruits5_fname_uk unique,                    -- UNIQUE
  grade varchar2(20) constraint fruits5_grade_nn not null,     -- NOT NULL
  fsize number        constraint fruits5_fsize_nn not null     -- NOT NULL
);
```

```
-- 확인 : 구조(해당 테이블)
desc fruits5;

-- 더미 데이터 삽입
insert into fruits5 (fid, fname, grade, fsize) values (1, 'apple',  'A', 10);
insert into fruits5 (fid, fname, grade, fsize) values (2, 'banana', 'B', 12);
insert into fruits5 (fid, fname, grade, fsize) values (3, 'grape',  'A',  8);
commit;

-- 확인: 데이터
select * from fruits5 order by fid;

/*
  PK: 줄마다 주민등록번호 같은 것(겹치면 X, 빈값 X)
  NOT NULL: 빈칸 금지
  UNIQUE: 서로 달라야 함
  순서: 만들고 → 넣고 → 확인 끝!
*/
```

무결성과 제약(Constraints)

무결성과 제약(Constraints)

- **데이터 무결성**: 정확·일관·유효 상태를 유지하는 성질.
- **PK (Primary Key)**: **유일 + NULL불가**로 한 행을 식별(테이블당 1개).
- **FK (Foreign Key)**: 다른 테이블의 PK/UNIQUE를 **참조**해 관계 보장.
- **UNIQUE**: 고유한 값 **중복 불가**(NULL은 대부분 허용).
- **NOT NULL**: **빈 값 금지**.
- **CHECK**: 값의 조건(범위/패턴 등) 강제.
- **참조 무결성**: FK가 항상 **존재하는 부모 키**만 가리키게 하는 규칙.
- **개체 무결성**: 각 행은 PK로 **반드시 구분 가능**해야 함.

📌구문 중간정리 (JS기준 메서드)

1️⃣ SELECT 관련

- `select` → 조회할 컬럼 지정
- `from` → 데이터 불러올 테이블 지정
- `where` → 조건 지정 (행 필터링)
- `desc 테이블명` → 테이블 구조 확인
- `order by 컬럼 [asc|desc]` → 정렬
- `group by 컬럼` → 집계 함수와 함께 그룹핑
- `having` → 그룹핑 후 조건 지정

2️⃣ 조건 연산자

- `=` , `!=` , `>` , `<` , `>=` , `<=` → 비교
- `between a and b` → 범위 지정
- `in (값1, 값2, ...)` → 값 목록 포함 여부
- `like '패턴'` → 문자열 패턴 검색 (`%` = 0개 이상, `_` = 1개 문자)
- `is null` / `is not null` → null 체크
- `exists (서브쿼리)` → 존재 여부

3️⃣ 집계 함수

- `count()` → 행 수
- `sum()` → 합계
- `avg()` → 평균
- `min()` → 최소값
- `max()` → 최대값

4️⃣ 조인 관련

- `join` → 두 테이블 연결
  - `inner join` → 조건 일치 행만
  - `left join` → 왼쪽 테이블 기준 모두, 오른쪽 없는 행은 null

- `right join` → 오른쪽 기준 모두
- `full join` → 양쪽 테이블 모두

5️⃣ 데이터 조작

- `insert into 테이블명 values(...)` → 행 추가
- `update 테이블명 set 컬럼=값 where 조건` → 수정
- `delete from 테이블명 where 조건` → 삭제

6️⃣ 컬럼/테이블 별칭

- `as 별칭` → 컬럼명/테이블명 변경

7️⃣ 기타

- `distinct` → 중복 제거
- `union` / `union all` → 여러 쿼리 결과 합치기
- `subquery` → select 안에 select (단일행/다중행)

💡테이블.sql (실습10)

```
/*
 #테이블 복사하기
 - create table 테이블 명 as (서브쿼리는 select 밖에 없다.)
 - 테이블 복사시 제약조건 not null은 자동 복사
 */


select * from employees;

drop table emp2;  -- 혹시 모를 삭제
create table emp2 as (select * from employees); -- employees table을 emp2에 전체 복사
select * from emp2;

desc emp2;

-- 제약조건 확인
```

```
select * from user_constraints where table_name = 'EMPLOYEES';
select * from user_constraints where table_name = 'EMP2';

/*
emp2에는 데이터와 컬럼만 존재하고,
user_constraints에서 확인하면 PK/FK/UNIQUE 제약조건이 없다.
*/

select * from tab where tname = 'FRUITS6'; -- 기존 만들었던

-- 테이블 복사  FRUITS6 생성
create table frutis66 as (select * from fruits6);
select * from frutis66;

commit;

-- candy  테이블 생성
create table candy (
    color varchar2(20)
    constraint candy_color_nn not null
);

select * from candy;

-- 컬럼 추가
alter table candy
add (
    brand varchar2(30)
    constraint candy_brand_nn not null,
    country_code varchar2(8)
    constraint candy_cc_nn not null
);

-- unique 제약조건 추가 (brand 중복 금지)
alter table candy add constraint candy_brand_nk unique (brand);

-- 컬럼 이름을 바꾸는 문장
alter table candy rename column country_code to code;
```

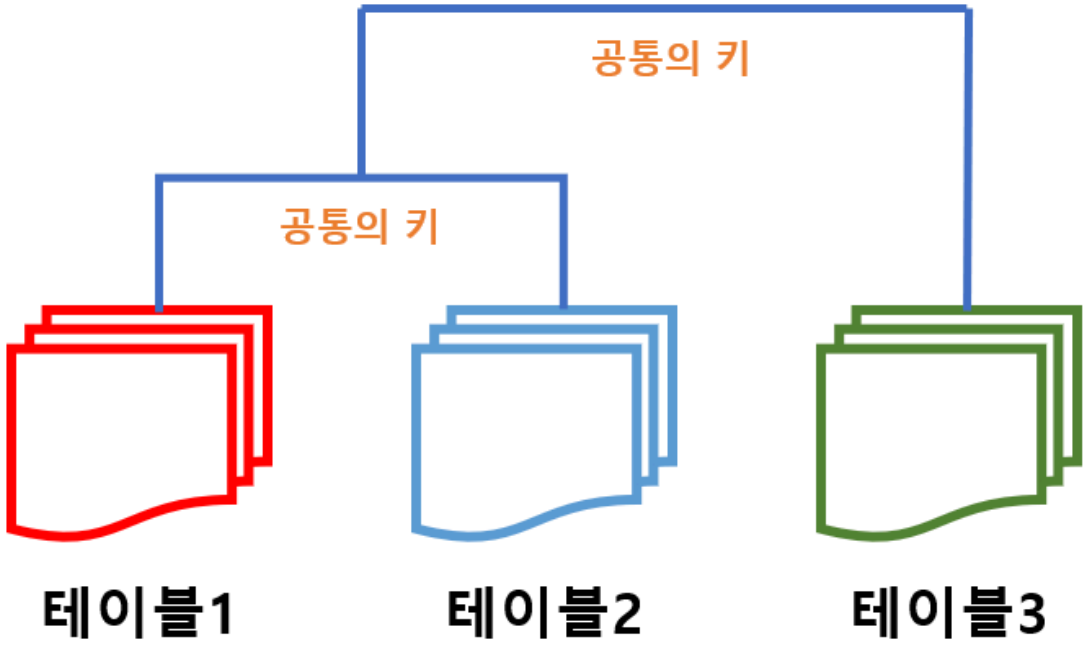
```
/*
 #테이블 컬럼 타입 변경
  - alter table 테이블 명 modify (컬렴명 컬럼타입,...);
  alter table candy modify (code varchar2(4));
*/

alter table candy modify (code varchar2(4));
desc candy; -- 타입 확인

/*
 #테이블 제약조건 삭제
 - alter table 테이블명 drop constraint 조건명;
*/

create table kd_acadmy(
 id varchar2(20)
    constraint kd_id_nn NOT NULL -- nn 조건 생성
);

select * from kd_acadmy;
select * from user_constraints where table_name = 'KD_ACADMY';

commit;

alter table kd_acadmy drop constraint kd_id_nn; -- 해당 테이블 제약 조건 삭제 (drop)
```

💡서브쿼리.sql (실습11)

```
/*
    # 서브쿼리
    -하나의 select 문 내부에 포함된 또하나의 select문
    -서브 쿼리를 포함하고 있는 쿼리를 '메인쿼리' 라고함
    -서브 쿼리가 먼저 실행
    -실행 결과에 따라 단일행 서브쿼리 , 다중행 서브쿼리로 분류됨
```

```
    # 단일행 서브쿼리
    -서브쿼리의 실행결과 단 하나의 행인 서브쿼리
    -단일 값끼리 비교하는 연산들을 사용할수있음
    -예) = , > ,<

    # 다중행 서브쿼리
    -서브쿼리의 실행 결과가 2행이상인 서브쿼리
    -다중행 연산자와 함께 사용해야 함
    -IN,SOME,ANY,ALL,EXISTS
*/

/* IN : IN 뒤에 나오는 여러값들 중 해당 값이 포함되어 있으면 TRUE  */
select * from jobs;

/*employees 테이블의 first_name 컬럼값이 'Peter' 인 job_id 값이 하나라도 일치한것만 뽑아냄*/
select * from jobs where job_id
IN(select job_id from employees where first_name ='Peter');


/* ANY ,SOME :뒤에 나오는 여러값들 중 하나 이상의 해당 조건을 만족시키면 트루 */
/*job_id가 IT_PROG를 가진 직원들의 salary(급여)를 모두 뽑음*/
select * from employees
where salary > SOME (select salary from employees where job_id ='IT_PROG');



/* ALL : 뒤에 나오는 여러값들의 모든 조건을 만족시키면 True */
select * from employees
where salary < ALL(select salary from employees where job_id = 'IT_PROG');
```

💡조인.sql (실습12)




```
/*

 #테이블 join (결과를 불러오는)
 - 기본키와 외래키로 관계가 형성되어 있는 테이블들의 정보를 종합하여 조회하는 것
 - 2개 이상의 테이블의 데이터를 한번 테이블에 가져올 수 잇음

 # 기본키 = Primary Key = PK
 - 한 테이블에서 하나의 행을 유일하게 구분할 수 있는 컬럼
 - 하나의 테이블의 하나의 기본키가 존재 = UNIQUE (유니크)
 - 기본키 설정 컬럼에는 NULL을 허용하지 않음 = NOT NULL

 #외래키 = Foreign Key = FK
 - 다른 테이블에는 기본키지만, 해당 테이블에서는 값이 중복인 컬럼
 - 어떤 테이블의 기본키가 다른 테이블의 외래키로 설정되면 두 테이블간에는 관계가 형성
 - (참고) 외래키로 설정된 컬럼에는 참조하는 테이블의 해당 컬럼에 존재하는 값만 추가 기능

 #references
 references는 다른 테이블의 기본키(pk)나 유니크 컬럼을 참조해서 연동(참조 무결성)을
 걸어주는 제약조건이다. 쉽게 말해서 한 테이블의 값이 다른 테이블에 반드시 존재하도
```

```
록
 강제하는것.

*/

select * from employees;
desc employees;

select * from user_constraints where table_name = 'EMPLOYEES'; -- 제약조건 확인
select * from departments;

desc departments;
select * from user_constraints where table_name = 'DEPARTMENTS'; -- 제약조건 확인

/*
 CROSS JOIN
 - 아무 의미 없는 조인 = 조인에 사용하는 테이블들의 데이터를 조합하여
   나올 수 있는 모든 경우의 수를 출력하는 조인
*/

-- employees, departments의 row수 = 총 컬럼수 = 레코드셋 갯수를 알아내기
select count(*) from employees; -- row 107개
select count(*) from departments; -- row 27개

select * from employees, departments; -- 107 * 27

/*
 EQUI JOIN
 - 두 테이블에서 서로 동일한 값을 지닌 컬럼(기본키,외래키)을 이용, CROSS JOIN에서 의미있는 데이터만 걸러내는 JOIN
 - 두 테이블에서 함께 사용할 때 같은 이름의 컬럼이 있다면, 앞에 테이블 이름을 (반드시)명기해야 함
*/

select * from employees;
select * from departments;
```

```
select * from locations;

-- 직원테이블과 부서테이블을 조인하여 department_id가 같은 행만 가져옴.
select * from employees a1, departments a2
where a1.department_id = a2.department_id;


-- 직원테이블과 부서테이블을 조인하여 사원의 이름과 부서명 조회
select a1.first_name, a2.department_name
from employees a1, departments a2
where a1.department_id = a2.department_id;


-- 실습. 직원테이블과 부서테이블 조인하여 직원번호 , 이름 , 부서명 조회 => 별칭사용
select
     emp.employee_id as 사원번호,
     emp.first_name as 이름,
     dept.department_name as 부서이름
from employees emp, departments dept
where emp.department_id = dept.department_id;

-- 실습. 직원테이블과 부서테이블 조인하여 직원번호 , 이름 , 근무지역번호 조회 => 별칭사용
select
    a1.employee_id as 직원번호,
    a1.first_name as 이름,
    a2.location_id as 근무지역번호
from employees a1, departments a2
where a1.department_id = a2.department_id;

-- 근무지역번호가 1700인 곳에서 근무하는 직원들의 직원번호, 이름, 직무를 조회
select
   a1.employee_id as 직원번호,
   a1.first_name as 이름,
   a1.job_id as 직무,
   a2.location_id as 근무지역번호
from employees a1, departments a2
```

```
where a1.department_id = a2.department_id and a2.location_id  = 1700;


-- 실습. Sales부서에 근무하는 직원들의 직원아이디, 이름, 근무지역번호, 부서명 조회
select emp.employee_id as 직원번호,
       emp.first_name as 이름,
       dept.location_id as 근무지역번호,
       dept.department_name as 부서명
from employees emp, departments dept
where emp.department_id = dept.department_id
and dept.department_name = 'Sales';


-- 실습. 2002년에 입사한 직원들의 직원번호, 이름, 입사일, 근무부서 조회 | (정렬) 입사일순으로 정렬

select emp.employee_id as 직원번호,
       emp.first_name as 이름,
       emp.hire_date as 입사일,
       dept.department_name as 부서명
from employees emp, departments dept
where emp.department_id = dept.department_id
and emp.hire_date between '02/01/01' and '02/12/31'
order by emp.hire_date;


/* # 3개 테이블 조인 */
select * from employees;
select * from departments; -- location_id(fk)
select * from jobs; -- job_id, job_title, 연봉범위
select * from locations; -- country_id(fk), city(도시이름)

-- 직무가 IT_PROG인 직원들의 이름/부서명/도시이름/직무 조회
select
   employees.first_name,
   departments.department_id,
   locations.city,
   employees.job_id
```

```
from employees, departments, locations
where employees.department_id = departments.department_id
AND departments.location_id = locations.location_id
AND job_id = 'IT_PROG';

-- 실습. 사는 도시가 'Seattle'인 직원의 이름/직급/급여/사는도시 조회 | 테이블 4개 조인
select
 e.first_name,
 j.job_title,
 e.salary*(1+nvl(e.commission_pct,0)) as 급여,
 l.city
from employees e, jobs j, departments d, locations l
where d.department_Id = e.department_id
and l.location_id = d.location_id
and e.job_Id = j.job_id
and l.city = 'Seattle';
```

💡concat.sql (실습13)

```
/*
  concat 연산자 (||)
  - 문자열과 컬럼값을 합쳐서 하나의 문장처럼 출력
  - 형식: 컬럼 || '문자열' || 컬럼 || '문자열'
*/

select first_name || ' 사원의 담당업무는 ' || job_id || ' 입니다.'
from employees
where rownum <= 5; -- 상위 5명만 예시

/*
  distinct 연산자
  - select문 결과 중 중복된 값을 제거
  - 주로 중복된 데이터 확인, 유일한 값 조회 시 사용
*/
```

```
-- 중복 포함: 모든 부서번호 조회
select department_id from employees order by department_id;

-- 중복 제거: 유일한 부서번호만 조회
select distinct department_id from employees order by department_id;
```

concat (||) - 형식: 컬럼 || '문자열' || 컬럼 || '문자열'

select first_name || ' 사원의 담당업무는 ' || job_id || ' 입니다.'

distinct - 중복제거

💡ANSI조인.sql (실습14)

```
/*
 #_ANSI조인
  - 미국 국가 표준협회에서 정한 표현을 따르는 조인 문법
  - 조인 조건에 사용되는 컬럼명이 같으면 => using문을사용
  - 조인 조건에 on을 사용하므로 -> where 조건절과 구분하여 사용할 수 있는 좀더 편리한 가독성 표현법
*/

select * from employees;  -- employees 테이블 전체 조회
select * from departments;  -- departments 테이블 전체 조회
select * from locations;  -- locations 테이블 전체 조회
select * from jobs;  -- jobs 테이블 전체 조회

/* ANSI CROSS JOIN = 모든 조합 생성 (교차 조인) */
select * from employees cross join departments;  -- 직원과 부서 모든 조합 출력

/* EQUI 조인 = 기본키와 외래키로 조건 걸어 조인 */
select *
from employees emp, departments dept
where emp.department_id = dept.department_id;  -- employees와 departments를 department_id 기준으로 조인

/* ANSI INNER JOIN = EQUI 조인과 동일하게 동작 */
select  *
```

```
from employees emp
inner join departments dept
on emp.department_id = dept.department_id;  -- ON으로 조인 조건 명시

/* ANSI INNER JOIN + USING = 공통 컬럼명이 같을 때 사용 */
select department_id, first_name
from employees
inner join departments
using (department_id);  -- department_id 기준으로 조인, 결과 컬럼 중복 제거


-- 실습. 모든 직원들의 직원번호/이름/부서이름 조회 (inner join 사용)
select
 employee_id as 직원번호,  -- 직원 ID 컬럼
 first_name as 이름,       -- 직원 이름
 department_name as 부서이름  -- 부서 이름
from employees
inner join departments
using (department_id);  -- department_id 기준으로 조인



-- 3개 테이블 조인 = IT_PROG 직무 직원들의 이름/부서/도시 조회
select
 last_name as 성,         -- 직원 성
 first_name as 이름,       -- 직원 이름
 department_name as 부서이름, -- 부서 이름
 city as 도시              -- 근무 도시
from employees
inner join departments using (department_id)  -- employees ↔ department
s 조인
inner join locations using (location_id)      -- departments ↔ locations 조인
where job_id = 'IT_PROG';  -- 직무가 IT_PROG인 직원 필터링


-- 실습. Seattle 근무 직원들의 이름/직책/급여 조회 (같은결과)
select
    e.first_name as 이름,  -- 직원 이름
```

```
    j.job_title as 직책,   -- 직무 제목
    e.salary as 급여,      -- 급여
    l.city as 시티         -- 근무 도시
from
    employees e,           -- 직원 테이블
    departments d,         -- 부서 테이블
    locations l,           -- 위치 테이블
    jobs j                 -- 직무 테이블
where
    e.department_id = d.department_id  -- 직원 ↔ 부서 조인
    and d.location_id = l.location_id  -- 부서 ↔ 위치 조인
    and e.job_id = j.job_id            -- 직원 ↔ 직무 조인
    and l.city = 'Seattle';            -- Seattle 근무 필터링

-- ANSI JOIN + USING 방식 (같은 결과)
select
    e.first_name as 이름,   -- 직원 이름
    j.job_title as 직책,    -- 직무 제목
    e.salary as 급여,       -- 급여
    l.city as 시티          -- 근무 도시
from employees e
inner join departments d using (department_id)  -- employees ↔ departme
nts 조인
inner join locations l using (location_id)      -- departments ↔ locations 조인
inner join jobs j using (job_id)                -- employees ↔ jobs 조인
where l.city = 'Seattle';                       -- Seattle 근무 필터링


select * from employees;
select * from departments;
select * from jobs;

-- 실습. job_title(직책)이 Stock Manager인 직원들의 전화번호/직책을 조회 = ANS
I 조인 사용
select
    e.phone_number,
    j.job_title
from  employees e
```

```
inner join jobs j using(job_id)
where j.job_title = 'Stock Manager';



select * from user_constraints where table_name = 'EMPLOYEES';
select * from user_constraints where table_name = 'JOBS';
select * from user_constraints where table_name = 'DEPARTMENTS;

-- 실습. 월급을 최대 월급 이상으로 받는 직원들이 속한 부서를 조회. 단, 부서명의 중복 제거 = ANSI 조인
-- max  inner join distinct jobs_title

select * from employees;
select * from departments;
select * from jobs;



select
   DISTINCT department_name as 부서명
from
   employees
   inner join jobs using (job_id)
   inner join departments using (department_id)
where
   max_salary <= salary * (1 + nvl(commission_pct,0));
```

JOIN 종류 정리

1. 기본(옛날) 스타일 JOIN

```
SELECT e.employee_id, e.name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

- `FROM 테이블1, 테이블2` + `WHERE` 절에서 조건으로 연결

- 예전 SQL 방식, 지금은 잘 안 씀
- 가독성이 떨어지고, LEFT/RIGHT OUTER JOIN 같은 경우 표현이 복잡함

---

2. ANSI 표준 JOIN (Modern JOIN)

2-1. INNER JOIN

```
SELECT e.employee_id, e.name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

- `INNER JOIN`은 **두 테이블에서 조건에 맞는 데이터만 가져옴**
- `ON` 절로 연결 조건을 명확히 작성
- 가독성 좋고, OUTER JOIN, CROSS JOIN 등 확장성이 좋음
- 현대 SQL에서는 ANSI JOIN을 거의 사용

2-2. LEFT / RIGHT / FULL OUTER JOIN

```
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;
```

- LEFT JOIN → 왼쪽 테이블 기준 모두 가져오고, 조건 없는 오른쪽은 NULL
- RIGHT JOIN → 오른쪽 기준
- FULL OUTER → 양쪽 모두 포함

---

3. INNER JOIN vs 그냥 JOIN

```
SELECT ...
FROM employees e
JOIN departments d
ON e.department_id = d.department_id;
```

- `JOIN`만 쓰면 기본적으로 **INNER JOIN과 동일**
- 즉, `INNER` 생략 가능하지만 명시하는 게 가독성 좋음

👉핵심 요약

구분	문법 예	특징	쓰임새
옛날 JOIN	`FROM a, b WHERE a.id = b.id`	오래됨, 가독성 낮음	소규모 간단 쿼리, 레거시 시스템
ANSI JOIN	`FROM a INNER JOIN b ON a.id = b.id`	명확, 현대적, 가독성 좋음	표준, 유지보수 쉬움
INNER JOIN	`INNER JOIN` 혹은 `JOIN`	같은 의미	두 테이블 모두 조건 일치 데이터 필요할 때
LEFT/RIGHT/FULL JOIN	`LEFT JOIN`, etc	NULL 포함 가능	한쪽 테이블 기준 모두 포함 필요할 때

✨ 결론

- 결과가 같은 경우가 많지만, **문법과 가독성/확장성**이 다름
- 지금은 **ANSI JOIN 표기법이 권장**됨
- `JOIN` = `INNER JOIN`, 생략 가능
- OUTER JOIN은 구식 JOIN으로 표현하기 어렵기 때문에 ANSI JOIN 필요

3. USING (사용)

`USING`은 **ANSI JOIN 문법에서 JOIN할 때 공통 컬럼명을 간단히 지정**할 때 쓰는 방법이다.

```
SELECT e.employee_id, e.name, d.department_name
FROM employees e
INNER JOIN departments d
USING (department_id);
```

- `ON e.department_id = d.department_id` 대신

  `USING(department_id)` 이렇게 쓸 수 있음
- **조건 컬럼명이 양쪽 테이블에서 동일할 때만 사용 가능**
- 결과에 `department_id` 컬럼은 **한 번만 나타남**

- 가독성이 조금 더 깔끔해짐

4. ON vs USING 비교

```
-- ON 사용
SELECT e.employee_id, e.name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;

-- USING 사용
SELECT e.employee_id, e.name, d.department_name
FROM employees e
INNER JOIN departments d
USING (department_id);
```

- 결과는 거의 같음
- 다만 USING은 컬럼명이 **양쪽 동일해야** 함
- ON은 컬럼명이 달라도 매칭 가능 (`e.dept_id = d.id` 이런 경우)

👉핵심 요약

- `USING` = 같은 이름 컬럼끼리만 JOIN할 때 간편하게 쓰는 방법
- `ON` = 조건을 자유롭게 지정 가능
- 둘 다 **INNER, LEFT, RIGHT JOIN** 어디든 사용 가능

5. CROSS JOIN

```
SELECT *
FROM employees e
CROSS JOIN departments d;
```

- *모든 조합(Cartesian Product)**을 만들어냄
- employees 3명 × departments 3개 → 9행 생성
- 조건 없이 그냥 **모든 조합**

- 거의 잘 안 쓰고, 일부 통계/테스트용으로만 사용

---

6. EQUI JOIN

- 말 그대로 **같다(=) 조건으로 조인**하는 것
- 보통 `ON e.department_id = d.department_id` 처럼 사용
- ANSI JOIN에서는 **INNER JOIN + ON =** 형태가 EQUI JOIN
- 예시:

```
SELECT e.employee_id, e.name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;  -- 이게 EQUI JOIN
```

- `=` 외에 `<`, `>` 조건 쓰면 **NON-EQUI JOIN**이라고 함

💥 JOIN 종류 한눈에 정리

JOIN 종류	의미	예시	특징
INNER JOIN	조건 일치하는 데이터만	ON, USING	기본적이고 가장 많이 사용
LEFT/RIGHT/FULL OUTER JOIN	한쪽 테이블 기준 모두, 없는 건 NULL	LEFT JOIN	데이터 누락 방지용
CROSS JOIN	모든 조합	CROSS JOIN	조건 없이 조합, 거의 안씀
EQUI JOIN	= 조건 조인	INNER JOIN ... ON a.id = b.id	INNER JOIN 대부분 이 방식
NON-EQUI JOIN	<, >, BETWEEN 등 조건	ON a.salary > b.salary	조건이 등호가 아닌 경우

💡시퀀스.sql (실습15)

1. 시퀀스란?

- **자동 증가 숫자를 만들어 주는 객체**
- 주로 **기본키(PK)** 등 유니크 값으로 쓰려고 만듦
- 호출할 때마다 지정한 증가값만큼 **자동으로 증가**
- Oracle에서는 `NEXTVAL`과 `CURRVAL`로 값을 가져와 사용함

예시: `kd_seq.NEXTVAL` → 다음 번호 가져오기, `kd_seq.CURRVAL` → 마지막 가져온 번호 확인할 수 있다.

---

2. 쿼리 해석

```
-- 시퀀스 만들기
create sequence kd_seq increment by 1 start with 1;
-- kd_seq라는 시퀀스 생성하고 호출할때마다 1씩 증가 start는 1
```

- `kd_seq`라는 시퀀스 생성
- `start with 1` → 1부터 시작
- `increment by 1` → 호출할 때마다 1씩 증가

```
-- 시퀀스 확인 / 마찬가지로 '대문자' 조회
select * from user_sequences where sequence_name = 'KD_SEQ';
```

- **현재 스키마에 있는 시퀀스 중 이름이 KD_SEQ인 것** 조회
- 생성한 시퀀스 정보(현재값, 증가값, 최대값, 최소값 등)를 볼 수 있음

```
select * from all_sequences;
```

- **모든 시퀀스 정보**를 조회
- 다른 스키마까지 포함하여 볼 수 있음

---

3. 사용 예시

```
INSERT INTO employees(employee_id, first_name) VALUES (kd_seq.NEXTV
AL, '민수');
-- 테이블 삽입하는데 직원에다가 kd_seq 다음 벨류에 민수 넣어달라는 내용
```

- `kd_seq.NEXTVAL` 호출 → 1, `employee_id`에 자동 입력
- 다음 호출 시 2, 3 … 순차적으로 증가

```
/*
    #시퀀스
    - 시퀀스는 자동 증가 번호 생성기이다.
    - 주로 고유 식별자(ID)를 만들어야 할 때 사용되며 자동으로 숫자 넣어준다.
    - 조회시 : 시퀀스 이름은 대문자로 저장됨 (인지)

    create sequence 시퀀스명 [옵션명];
    [옵션명]
    [START WITH n]                - 시퀀스의 시작번호 설정
    [INCREMENT BY n]              - 시퀀스의 증가값 설정
    [MAXVALUE n | NOMAXVALUE]     - 최댓값 설정
    [MINVALUE n | NOMINVALUE]     - 최솟값 설정


    INSERT INTO employees(employee_id, first_name)
    VALUES (kd_seq.NEXTVAL, '민수');
    -- NEXTVAL 호출 시마다 시퀀스 번호가 자동으로 1씩 증가

    ※ 한줄요약
    employees 테이블에 새 행을 넣는데,
    직원 번호는 kd_seq 시퀀스가 자동으로 다음 번호를 주고, 이름은 '민수'로 넣어라

*/

-- 1. 전체 시퀀스 확인
SELECT * FROM all_sequences;

-- 2. 시퀀스 kd_seq 생성 - 시작은1 호출할때마다 1씩 증가
CREATE SEQUENCE kd_seq INCREMENT BY 1 START WITH 1;

-- 3. 전체 시퀀스 확인 = 'KD_SEQ' / 대문자 조회
SELECT * FROM all_sequences WHERE sequence_name = 'KD_SEQ';

-- 4. 테이블 생성 exeseq
CREATE TABLE exeseq(
```

```
   seq     NUMBER(4)    -- 자동번호
           CONSTRAINT exeseq_seq_pk    PRIMARY KEY,
   kind    VARCHAR2(30) -- 소속
           CONSTRAINT exeseq_kind_nn   NOT NULL,
   grade   CHAR(2)      -- 등급
           CONSTRAINT exeseq_grade_chk CHECK(grade IN('A','B','C','D')),
   gty     NUMBER(4)    -- 수량
           CONSTRAINT exeseq_gty_nn    NOT NULL,
   price   NUMBER(4)    -- 가격
           CONSTRAINT exeseq_price_nn  NOT NULL
);



-- 5. KD_EXESEQ 시퀀스 생성 1001 부터
CREATE SEQUENCE kd_exeseq  START WITH 1001 INCREMENT BY 1;

-- 5-1. KD_EXESEQ2 시퀀스 생성  1002 부터
CREATE SEQUENCE kd_exeseq2 START WITH 2001 INCREMENT BY 1;

-- 6. 시퀀스 추가한거 전체 확인 (KD_SEQ,KD_EXESEQ,KD_EXESEQ2)
SELECT * FROM all_sequences WHERE sequence_name IN ('KD_SEQ','KD_
EXESEQ', 'KD_EXESEQ2');

-- 7. 데이터 삽입 values에( kd_exeseq 시퀀스 사용 자동부여.NEXTVAL, '이름','등
급','수량','가격'  );
INSERT INTO exeseq VALUES(kd_exeseq.NEXTVAL,'KD','A',20,100);
INSERT INTO exeseq VALUES(kd_exeseq.NEXTVAL,'ACADEMY','B',10,300);

SELECT * FROM exeseq; --  exeseq 테이블에 삽입한 내용 잘 들어갔는지 한번 확
인

-- 7-1. 시퀀스 kd_exeseq2를 사용하여 한번더 삽입
INSERT INTO exeseq VALUES(kd_exeseq2.NEXTVAL,'ARE','C',10,400);

COMMIT;

-- 8. 데이터 확인 (kd_exeseq2 -  ARE이름으로 2001 부터시작)
```

```
SELECT * FROM exeseq ORDER BY seq;

-- 9. 테스트용 단일 컬럼 조회
SELECT * FROM dual;

-- EX) dual 쓰임새
SELECT 1+1 FROM dual;
-- 숫자 계산 결과 확인 : 2

-- SYSDATE(현재 날짜) 확인
SELECT SYSDATE FROM dual;

-- 시퀀스 kd_seq 값 확인  단일 컬럼만 보여줌
SELECT kd_seq.NEXTVAL FROM dual;

-- 시퀀스 현재 값 확인  CURRVAL - 마지막으로 생성된 값
SELECT kd_exeseq.CURRVAL FROM dual;  -- kd_exeseq 마지막 단일 행 조회
SELECT kd_exeseq2.CURRVAL FROM dual;  -- kd_exeseq2 마지막 단일 행 조
회

-- 10. 필요 시 테이블 및 시퀀스 삭제
-- 주석 처리하거나 필요 시 실행
DROP TABLE exeseq;
DROP SEQUENCE kd_seq;
DROP SEQUENCE kd_exeseq;
DROP SEQUENCE kd_exeseq2;


-- 시퀀스 들어간 테이블 조회
SELECT * FROM exeseq;
COMMIT;
```

💡트랜잭션(Transaction).sql (실습16)

```
/*
  # 트랜잭션 (Transaction)
```

```
  - 데이터 처리의 한 단위
  - 하나의 트랜잭션은 여러 명령어들로 이루어져 있음
  - 트랜잭션의 모든 과정이 정상적으로 완료되는 경우에만
    변경사항이 확정됨 (ALL or Nothing)
  - 트랜잭션을 관리하기 위한 명령어COMMIT , ROLLBACK, SAVEPOINT등이 있음
  - 하나의 트랜잭션은 마지막 실행 된 COMMIT, ROLLBACK 으로 부터
    새로운 커밋을 실행하기 전까지 수행하는 SQL문을 말함

  (예)
  오라클 DB = 개발자가 전달한 쿼리문(INSERT문, UPDATE문, DELETE문)은
  메모리상에서만 수행
  실제 영속성을 위한 작업을 하지 않음 = 하드디스크(물리적)에    전달 안함
  => 실수로 인해 데이터의 유실을 막기 위한 장치임
  => CREATE TABLE,

  drop table cafe_menu;
*/



create table cafe_menu(
  mid number(4)
      CONSTRAINT cm_mid_pk  PRIMARY KEY
);



alter table cafe_menu ADD(
    name varchar2(30)
         constraint cm_name_uk UNIQUE
         constraint cm_name_nn NOT NULL,

    price number(5)
          constraint cm_price_chk CHECK( price >= 10)
          constraint cm_price_nn NOT NULL,

    min_size varchar2(1)
            constraint cm_min_size_chk CHECK(min_size in('S','M','L'))
```

```
        constraint cm_min_size_nn NOT NULL,

    max_size varchar2(1)
        constraint cm_max_size_chk CHECK(max_size in('S','M','L'))
        constraint cm_max_size_nn NOT NULL
);
-- mid, name, price, min_size, max_size

-- 더미데이터 5개 삽입
insert into cafe_menu values(1, '아메리카노', 4000, 'S', 'L');
insert into cafe_menu values(2, '카페라떼', 4500, 'S', 'L');
insert into  cafe_menu values(3, '카푸치노', 4800, 'S', 'L');
insert into  cafe_menu values(4, '카페모카', 5000, 'S', 'L');
insert into  cafe_menu values(5, '에스프레소', 3500, 'S', 'M');

select * from cafe_menu; --테이블 확인

-- 하나의 트랜잭션이 실행하다 commit으로 커밋 = 물리적 하드웨어에 저장
commit;


-- ※ savepoint는 저장은 마지막 한 포인트만 저장됨
-- 실행문 : rollback to savepoint명

insert into cafe_menu values (6,'자몽에이드',6700,'M','L');

savepoint svp1;

insert into cafe_menu values (7,'녹차라떼' ,5800,'S','L');

savepoint svp2;

insert into cafe_menu values(8, '아이스크림', 5300 ,'S','L');

insert into cafe_menu values(100, '롤케이크', 19900 ,'S','M');

savepoint svp3;
```

```
rollback to svp2;
```

이쯤에서 헷갈릴만한 부분으로 넘어가야할것은 부분을 정리해보았다.

- **INSERT INTO**
  - **기존 테이블에 새 데이터 행(row)을 추가**
  - 테이블 구조는 그대로 두고, **데이터 내용만 추가**
  - 예: 새 메뉴, 새 직원, 새 주문 등
- **ALTER TABLE**
  - **테이블 구조 자체를 변경**
  - 컬럼 추가/삭제, 제약조건 추가/변경 등
  - 데이터는 그대로 두고, **테이블 틀(schema)만 변경**

💡function.sql (실습17)

```
/*

  # 함수
    - 오라클의 함수를 이용하면 테이블의 데이터를 가공하여 조회가 가능하다.

*/

-- DUAL 테이블 조회
SELECT * FROM dual;

-- ABS(n) : 절대값
SELECT abs(-123) FROM dual;

-- FLOOR(n) : 내림 - 마룻바닥
SELECT floor(123.1234) FROM dual;

-- CEIL(n) : 올림 - 천장
SELECT ceil(123.1234) FROM dual;
```

```
-- ROUND(n) : 반올림
SELECT round(123.1234) FROM dual;

-- MOD(n, m) : 나머지 계산
SELECT mod(17, 10) FROM dual;

-- TO_CHAR(값 [, 형식]) : 숫자나 날짜 데이터를 문자(String)로 변환
SELECT to_char(123) FROM dual;
SELECT TO_CHAR(12345) FROM dual;
-- 결과: '12345' (문자형으로 변환)

SELECT TO_CHAR(12345, '000000') FROM dual;
-- 결과: '012345' (형식 지정, 6자리로 맞춤)

-- SYSDATE 사용 예시: 현재 날짜를 문자열로 변환
SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI:SS') FROM dual;
-- 결과 예시: '2025-10-17 11:45:00'

-- TO_DATE() : 데이터를 날짜 타입으로 변환
SELECT TO_DATE('2025-10-17','YYYY-MM-DD') FROM dual;
-- 결과: DATE 타입으로 '2025-10-17'

-- TRIM() : 양쪽의 공백 제거
SELECT TRIM('   Hello World   ') AS trimmed_string FROM dual;
-- 결과: 'Hello World'
```

💡view.sql (실습18)

```
/*
   # 뷰(view)
     - 데이터베이스에서 제공하는 가상의 테이블
     - 뷰 사용시 복잡한 쿼리문을 대신할 수 있으므로 개발의 용의성을 가짐
     - 뷰를 만들때 사용한 쿼리문을 저장하는 것
     - 주로 복잡한 서브쿼리 사용시 다음에 다시 사용할 목적으로 사용

   (예) 여러 테이블을 조인한 복잡한 쿼리문의 결과를
```

```
        => 가상의 테이블 뷰에 저장하면, 그 이후부터 복잡한 쿼리를 쉽게 사용할 수 있음

   # 뷰 생성하기   - view 생성 구문에서의 AS는 *"이 뷰는 다음 쿼리 결과를 기반으로 한다"*라는 의미
      CREATE VIEW 뷰이름
      AS
      서브쿼리

   ※ 결론 테이블을 직접 생성안하고 가상테이블에 넣어보는것 ※

*/



/* # 뷰 만들기 */
-- 직원의 사원번호, 이름, 급여, 근무부서, 근무지역 = EQUI 조인 이용하여 작성
select * from employees;
select * from departments;

create view emp_dept_view -- 가상테이블 생성
AS
select
   e.employee_id,
   e.first_name,
   e.salary,
   d.department_id,
   d.location_id
from
   employees e, departments d
where
   e.department_id = d.department_id;

select * from emp_dept_view;

-- 뷰삭제
DROP VIEW emp_dept_view;

-- 시스템 카탈로그 조회 = 데이터 딕셔너리
```

```
select * from user_catalog;

-- 실습. 뷰 만들기
-- 사는 도시가 'Seattle'인 직원의 이름/직급/급여/사는도시 조회 = EQUI = 테이블 4
개 조인
select * from employees;
select * from departments;
select * from jobs;
select * from locations;
CREATE VIEW emp_dept_jobs_loc_view
AS
select
    e.first_name,
    j.job_title,
    e.salary*(1+nvl(commission_pct,0)) as 급여,
    l.city
from
    employees e,
    departments d,
    jobs j,
    locations l
where
    d.department_id = e.department_id
    and
    l.location_id = d.location_id
    and
    e.job_id = j.job_id
    and
    l.city = 'Seattle';
-- 뷰 조회
select * from emp_dept_jobs_loc_view;

-- 사는 도시가 'Seattle'인 직원의 이름/직급/급여/사는도시 조회 = EQUI = 테이블 4
개 조인

select * from employees;
select * from departments;
select * from jobs;
```

```
select * from locations;


SELECT
  e.first_name,
  j.job_title,
  e.salary,
  l.city
FROM employees e
INNER JOIN departments d
    ON e.department_id = d.department_id
INNER JOIN jobs j
    ON e.job_id = j.job_id
INNER JOIN locations l
    ON d.location_id = l.location_id
WHERE l.city = 'Seattle';
```

view 생성 구문에서의 AS는 **“이 뷰는 다음 쿼리 결과를 기반으로 한다”**라는 의미이다.

뷰 생성하기 - create view (해당 뷰명) AS (서브쿼리);

뷰 삭제 - drop view (해당 뷰명);
시스템 카탈로그 조회 : SELECT * FROM user_catalog;

💡마무리.sql (시험)

직원(emp)

직원코드(u_id)	가변문자(8)	기본키
이름(name)	가변문자(20)	널 값 허용안함
비밀번호(password)	가변문자(20)	널 값 허용안함
직위코드(position)	가변문자(4)	널 값 허용안함
근무지(workplace)	가변문자(20)	널 값 허용안함
집주소(address)	가변문자(50)	
생년월일(birth)	날짜	
입사일(regday)	날짜	
결혼기념일(wedday)	날짜	

급여(sal)

직원코드(u_id)	가변문자(8)	기본키
급여(salary)	정수(10)	널 값 허용안함
근무시작일(from_date)	날짜	
근무종료일(to_date)	날짜	

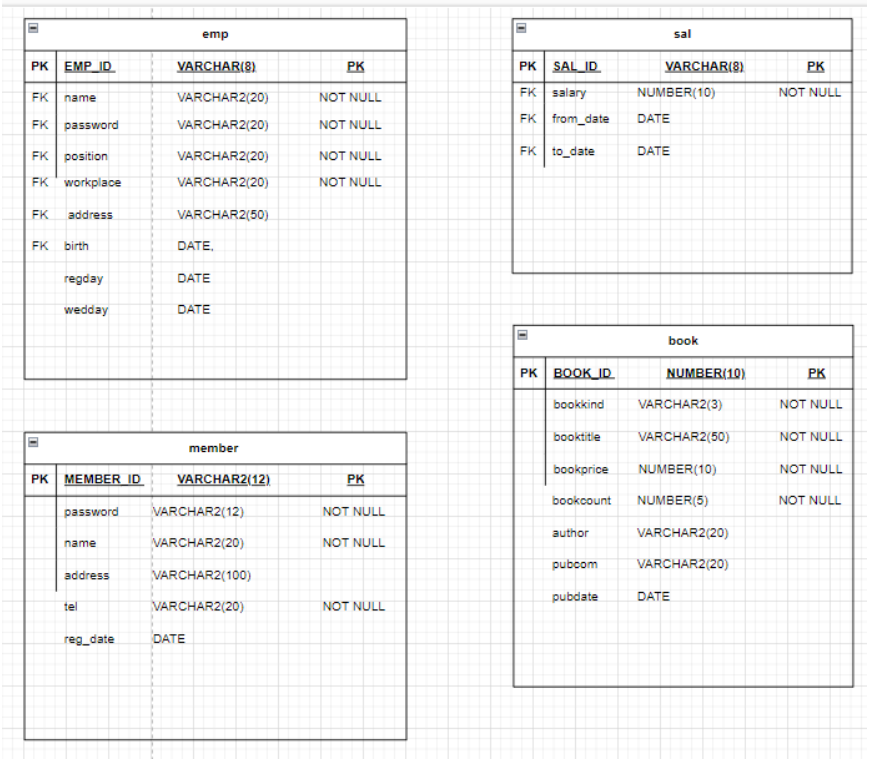
회원(member)

아이디(id)	가변문자(12)	기본키
비밀번호(password)	가변문자(12)	널 값 허용안함
회원명(name)	가변문자(20)	널 값 허용안함
주소(address)	가변문자(100)	
연락처(tel	가변문자(20)	널 값 허용안함
가입일(reg_date)	날짜	

도서(book)

도서코드(bookid)	숫자(10)	기본키
도서분류(bookkind)	가변문자(3)	널 값 허용안함
도서제목(booktitle)	가변문자(50)	널 값 허용안함
도서가격(bookprice)	숫자(10)	널 값 허용안함
도서수량(bookcount)	숫자(5)	널 값 허용안함
저자(author)	가변문자(20)	
출판사(pubcom)	가변문자(20)	
출판일(pubdate)	날짜	

오라클명령문법및실습_문제1.xlsx



emp

PK	EMP_ID	VARCHAR(8)	PK
FK	name	VARCHAR2(20)	NOT NULL
FK	password	VARCHAR2(20)	NOT NULL
FK	position	VARCHAR2(20)	NOT NULL
FK	workplace	VARCHAR2(20)	NOT NULL
FK	address	VARCHAR2(50)	
FK	birth	DATE,	
	regday	DATE	
	wedday	DATE	

sal

PK	SAL_ID	VARCHAR(8)	PK
FK	salary	NUMBER(10)	NOT NULL
FK	from_date	DATE	
FK	to_date	DATE	

book

PK	BOOK_ID	NUMBER(10)	PK
	bookkind	VARCHAR2(3)	NOT NULL
	booktitle	VARCHAR2(50)	NOT NULL
	bookprice	NUMBER(10)	NOT NULL
	bookcount	NUMBER(5)	NOT NULL
	author	VARCHAR2(20)	
	pubcom	VARCHAR2(20)	
	pubdate	DATE	

member

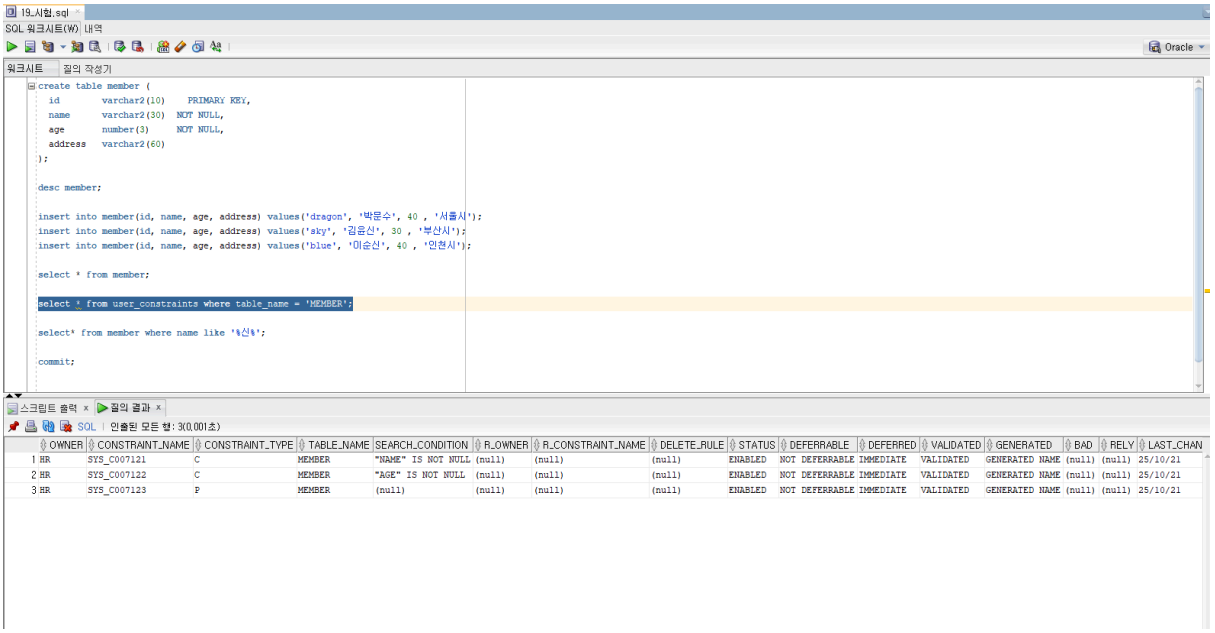
PK	MEMBER_ID	VARCHAR2(12)	PK
	password	VARCHAR2(12)	NOT NULL
	name	VARCHAR2(20)	NOT NULL
	address	VARCHAR2(100)	
	tel	VARCHAR2(20)	NOT NULL
	reg_date	DATE	

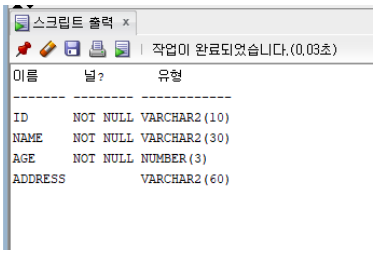
ERD.drawio

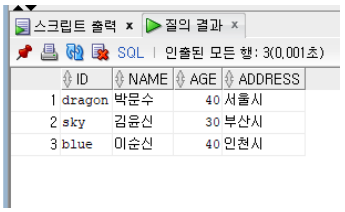
```
CREATE TABLE emp (
  u_id        CHAR(8)       PRIMARY KEY,
  name        VARCHAR2(20)  NOT NULL,
  password    VARCHAR2(100) NOT NULL,
  position    CHAR(4)       NOT NULL,
  workplace   VARCHAR2(20)  NOT NULL,
  address     VARCHAR2(50),
  birth       DATE,
  regday      DATE,
  wedday      DATE
);

CREATE TABLE sal (
```

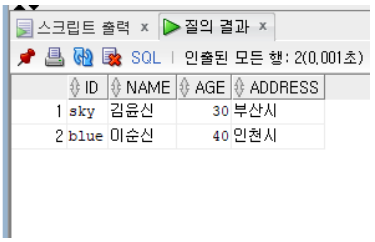
```
  u_id        CHAR(8)       PRIMARY KEY,
  salary      NUMBER(10)    NOT NULL,
  from_date   DATE,
  to_date     DATE,
  CONSTRAINT fk_sal_emp FOREIGN KEY (u_id) REFERENCES emp(u_id)
);

CREATE TABLE member (
  id          VARCHAR2(12)  PRIMARY KEY,
  password    VARCHAR2(100) NOT NULL,
  name        VARCHAR2(20)  NOT NULL,
  address     VARCHAR2(100),
  tel         VARCHAR2(20)  NOT NULL,
  reg_date    DATE
);

CREATE TABLE book (
  bookid      NUMBER(10)    PRIMARY KEY,
  bookkind    CHAR(3)       NOT NULL,
  booktitle   VARCHAR2(50)  NOT NULL,
  bookprice   NUMBER(10)    NOT NULL,
  bookcount   NUMBER(5)     NOT NULL,
  author      VARCHAR2(20),
  pubcom      VARCHAR2(20),
  pubdate     DATE
);
```

직접 ERD를 만들어 시각화 하고 SQL 쿼리문 작성해보는것까지 알아보았다.










```
-- 생성
create table member (
  id       varchar2(10)    PRIMARY KEY,
  name     varchar2(30)  NOT NULL,
  age      number(3)     NOT NULL,
  address  varchar2(60)
);

-- PK, FK 확인
desc member;

--삽입
insert into member(id, name, age, address) values('dragon', '박문수', 40 , '서울시');
insert into member(id, name, age, address) values('sky', '김윤신', 30 , '부산시');
```

```
insert into member(id, name, age, address) values('blue', '이순신', 40 , '인천시');

select * from member;


-- 제약조건 조회
select * from user_constraints where table_name = 'MEMBER';

-- 이름이 신이 들어간사람 찾기
select* from member where name like '%신%';

-- 영속성
commit;


-- 행 삭제
delete from member;

-- commit 시점으로 돌아가기
rollback;
```