0. Oracle 설치, SQLDeveloper 실행

1. **Oracle 21C XE Edtion 설치 및 2. SQLDeveloper 실행**
2. **Oracle 21C XE Edtion 설치**

**주의 : 오라클 설치시 다른 작업 금지**

**TCP : 1521**

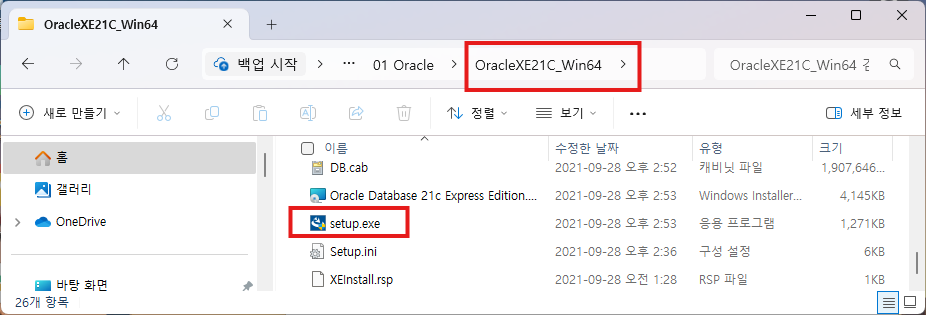
**열려있는 포트 확인 : netstat –na | more**

**1-1 두 파일을 다운 받아서 압축을 푼다.**

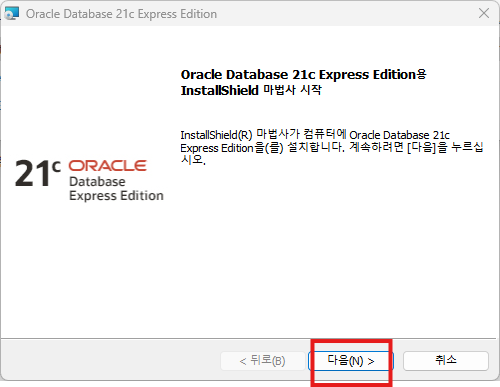
**OracleXE21C\_Win64.zip 🡸 Oracle Sever**

**sqldeveloper-x64.zip 🡸 Oracle Server 원격에서 관리 하는 client 관리툴**

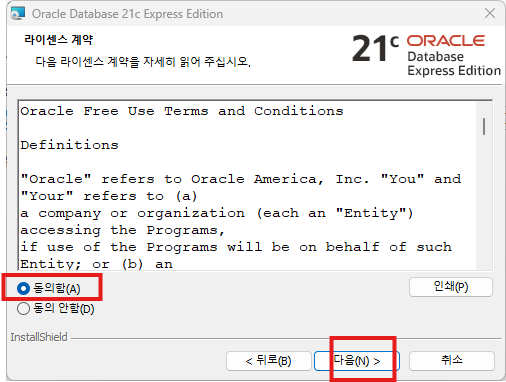
**1-2. 압축을 푼 OracleXE21C\_Win64 폴더에서 setup을 실행한다.**



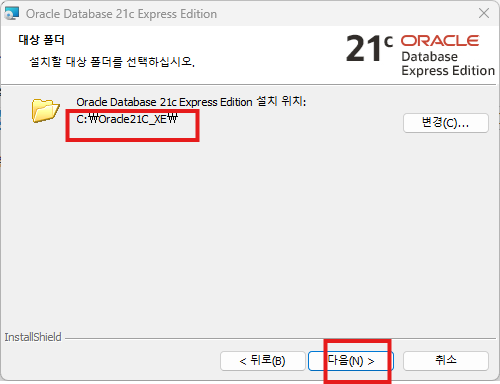
**1-3. [다음] 클릭**



**1-4. 라이센스 계약 : [동의함] 체크, [다음] 클릭**

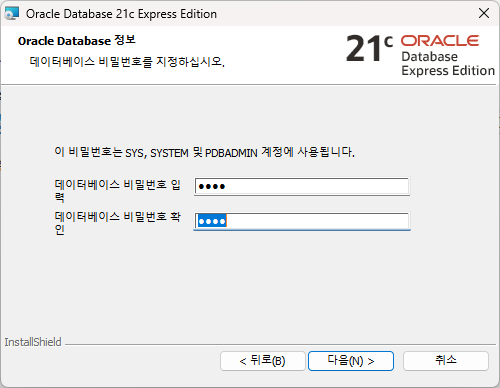


**1-5. 설치 폴더 경로 변경 함. ( C:\Oracle21C\_XE\ ) : [주의] - 설치 폴더 변경**



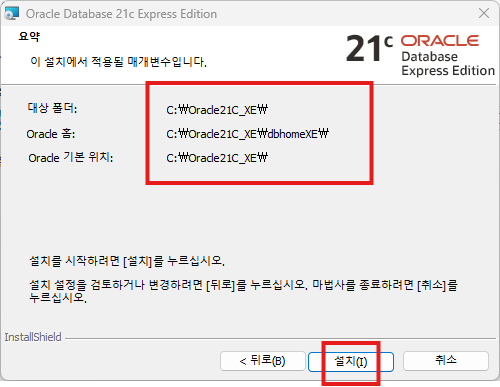
**1-6. DataBase SYS, SYSTEM, PDBADMIN 계정의 암호 부여 : 1234**

**오라클의 최고 관리자 계정의 암호 부여**

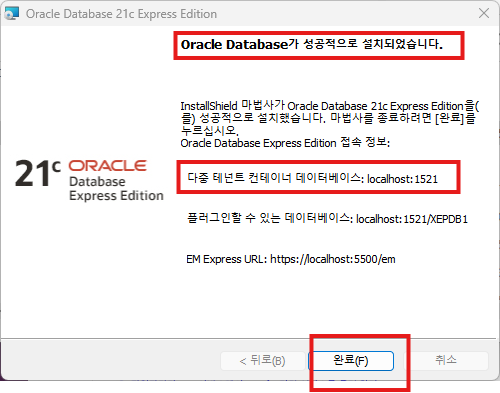


**1-7. 요약 정보 확인후 [설치] 클릭 : 시스템 사양에 따라 10분 ~ 30분 정도 걸림**

**주의 : 설치 완료시 까지 다른 작업 금지**

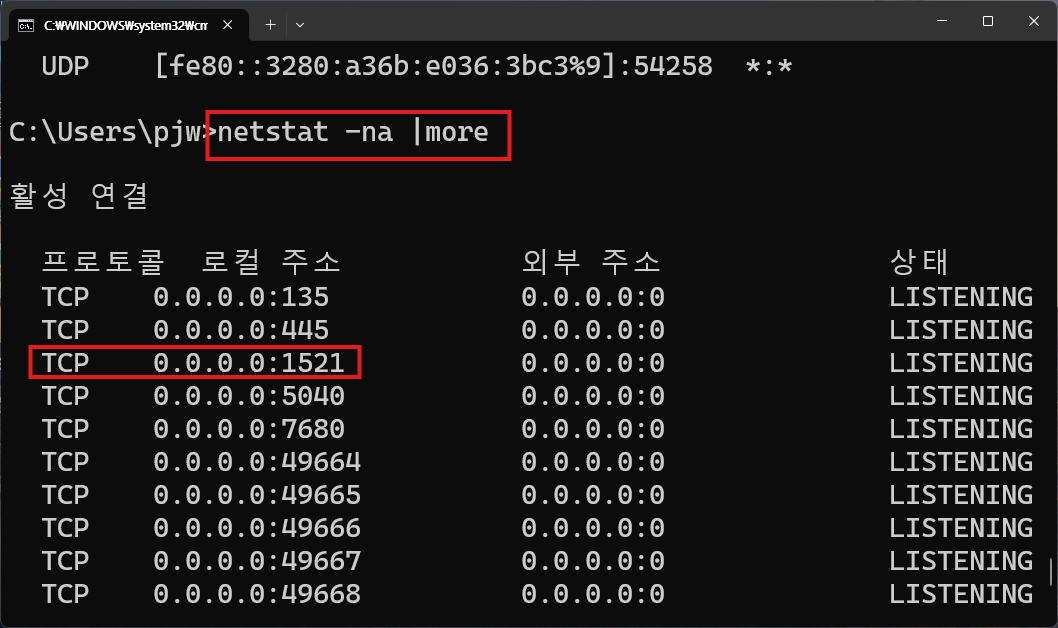


**1-8. 설치 완료. [Oracle Database가 성공적으로 설치 되었습니다.] 확인**

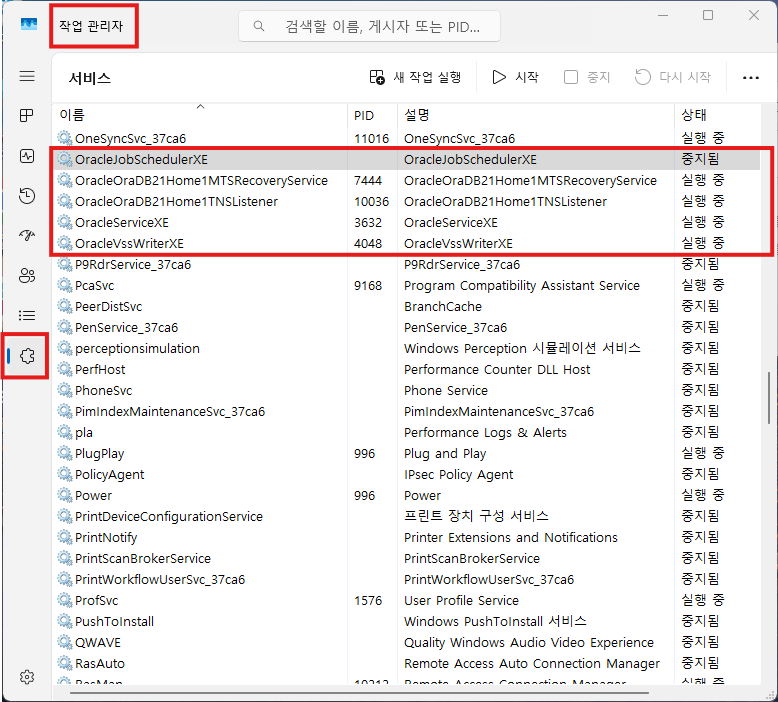


**1-9. 설치 완료 확인 , 명령프롬프트[cmd] 에서 명령어 입력, TCP 1521 이 존재하는지 확인**

**netstat -na | more**

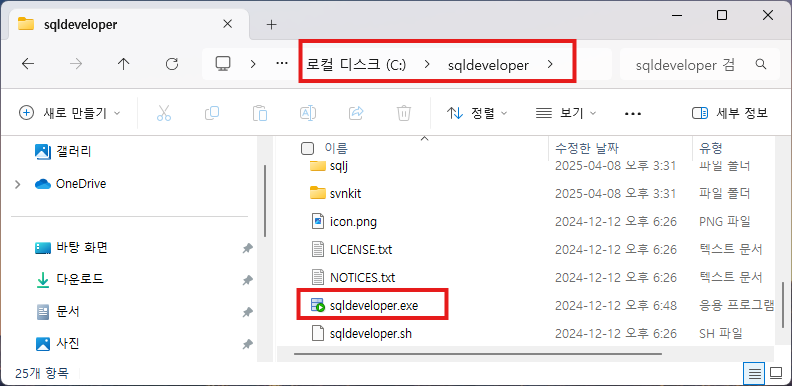


**1-10. 설치 완료 확인 - [작업관리자] [서비스] 에서 아래 서비스가 실행중인지 확인**



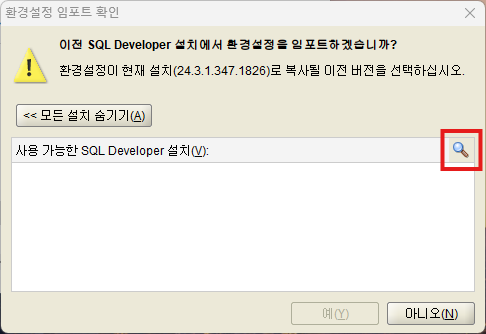
**2. SQLDeveloper 실행**

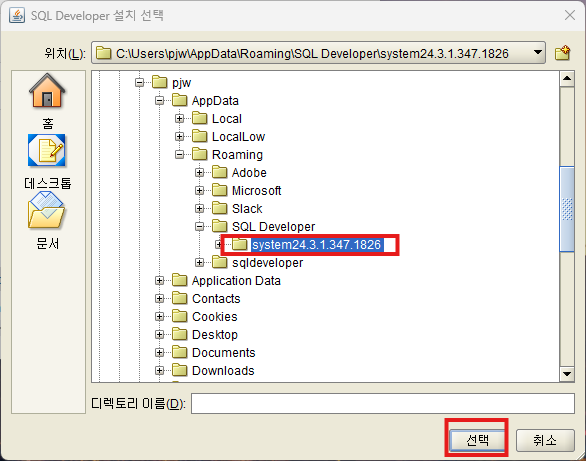
**2-1. Oracle Developer 실행 압축을 풀고 압축푼 폴더의 developer를 c:\developer\ 로 이동 후 sqldeveloper.exe 실행 : 주의 요함**

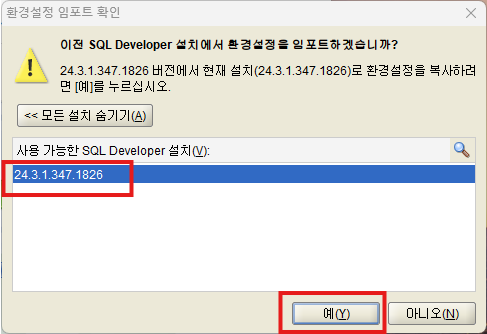


**2-2. 환경설정 임포트 확인 : [검색] 아이콘을 클릭 하여 아래를 선택한다.**

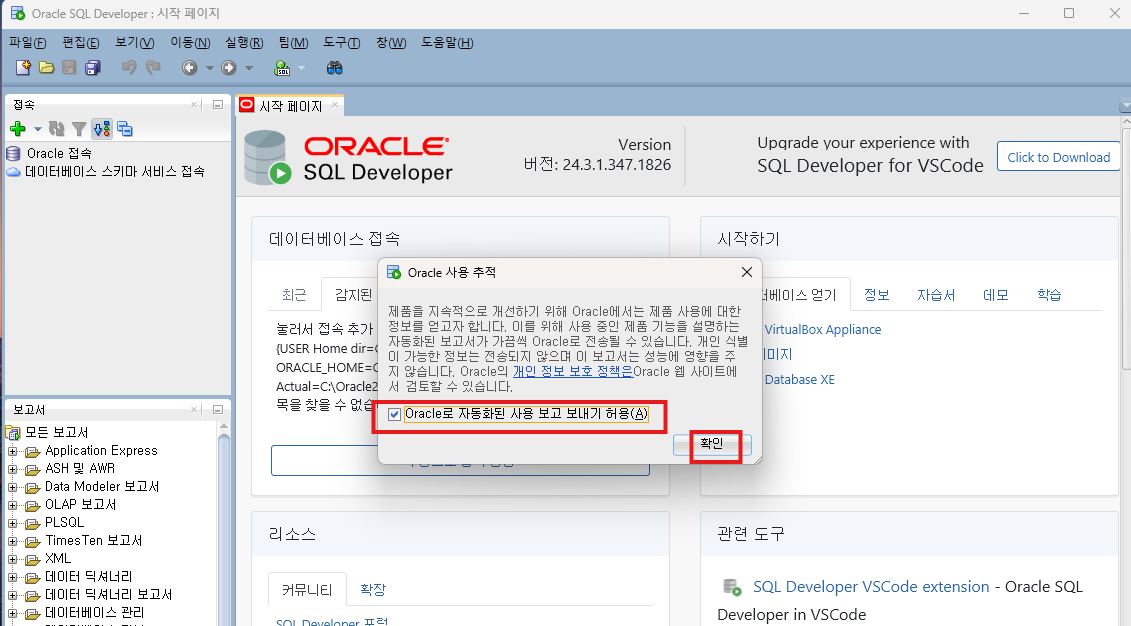
**주의 : 아래 내용 잘 선택 하여야 함. [처음 실행시에만 선택 해주면 됨.]**





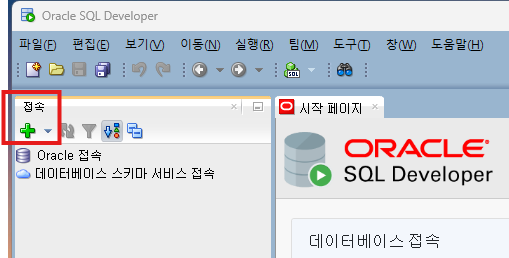


**2-3. Oracle SQL Developer 실행됨 , [Oracle 사용추적 창] 에서 [확인] 클릭**



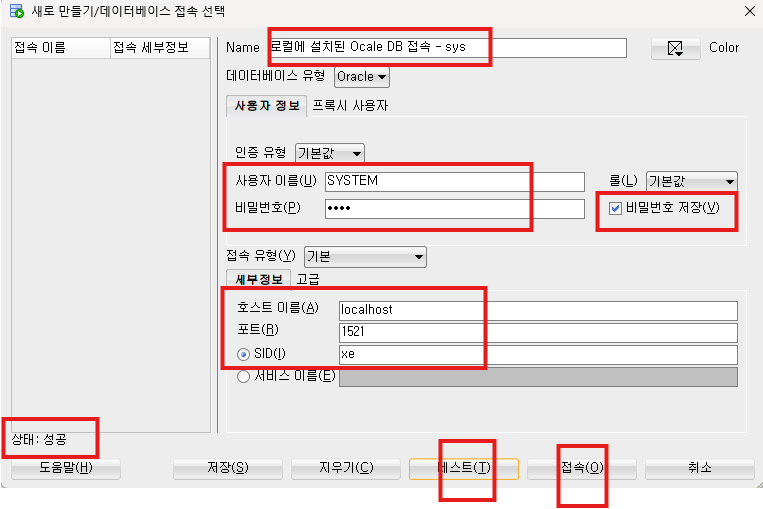
**2-4. Oracle SQL Devedoper 실행 완료 : 원격의 Oracle 서버를 관리 하는 Client 관리툴**

**2-5. [데이터베이스 접속] 생성 - 왼쪽 상단의 [접속 +] 클릭**

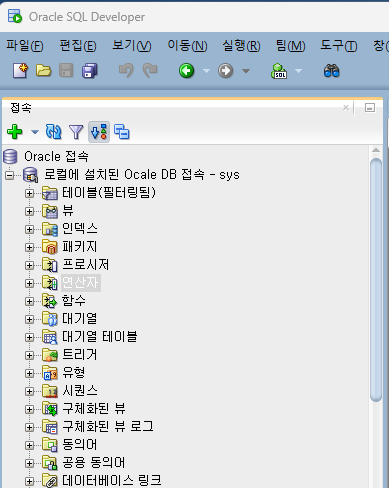


**2-6. 아래 내용 입력후 [테스트], [접속] 클릭**

**Name : 로컬에 설치된 Ocale DB 접속 – SYSTEM [ 오라클의 최고 관리자 계정 ]**



**2-7. 접속 완료 : 아래를 확장 해서 접속이 잘 된 것을 확인**



1. HR 계정 생성 - SYSTEM 계정 접속

1. HR 계정 생성 - SYSTEM 계정 접속

| -- SQL의 주석 : 한줄 주석  /\*  여러줄 주석  \*/  /\*  일반 계정을 생성후 일반 계정으로 접속 : 계정 생성, 권한 부여  \*/  -- system 계정으로 접속한 쿼리창에서만 계정을 생성 할 수 있다.  -- 1. HR 계정을 생성 : Oracle 12 버젼 이상 부터 계정 생성시 **C##계정명 ⇐ 공백주의, 공백없음**  -- 계정명 : HR 암호 : 1234  create user C##**HR** identified by 1234  default tablespace USERS  temporary tablespace TEMP;  -- 2. HR 계정에 권한을 부여하기 : connect : 접속 , resource : 객체 생성, 수정, 삭제  grant resource, connect to C##**HR**;  grant create view to C##**HR** ;  --3. HR 계정에게 테이블 스페이스에서 사용량 할당.  alter user C##**HR** quota unlimited on USERS;  --4. 계정 삭제  drop user C##HR cascade |
| --- |

**// 간단한 테이블 생성 및 테이블에 값넣어보기**

| /\*  CRUD : Create, Read, Update, Delete  (생성), (검색), (수정) (삭제)  \*/  -- 간단한 테이블 생성  Create Table member (  idx number not null ,  name varchar2(50) not null,  id varchar2(50) not null,  pw varchar2(50) not null,  addr varchar2(50) null  );  -- 검색(출력하라) : select 문  select \* from member ;  -- 입력 : 테이블에 값을 넣음. 반드시 commit; 실행  insert into member ( idx, name, id , pw, addr)  values (1 , '홍길동', 'pjw', '1234', '서울') ;  commit; -- DB에 영구히 저장하라  insert into member ( idx, name, id , pw, addr)  values (2 , '김길동', 'pjw1', '1234', '부산') ;  commit;  // 실습  -- C##HR2 계정을 생성하고, HR2 계정으로 접속, member 테이블 생성후 레코드 2개 넣기 |
| --- |

2. 실습용 샘플 스크립트 생성

2. 실습용 샘플 스크립트 생성

| **/\***  **테이블 삭제 (순서 주의)**  **DROP TABLE EMPLOYEE;**  **DROP TABLE DEPARTMENT;**  **DROP TABLE SALGRADE;**  **\*/**  **/\* 테이블 생성 : DEPARTMENT, EMPLOYEE, SALGRADE \*/**  **CREATE TABLE DEPARTMENT (**  **DNO NUMBER(2) CONSTRAINT PK\_DEPARTMENT\_DNO PRIMARY KEY,**  **DNAME VARCHAR2(14),**  **LOC VARCHAR2(13)**  **);**  **CREATE TABLE EMPLOYEE (**  **ENO NUMBER(4) CONSTRAINT PK\_EMPLOYEE\_ENO PRIMARY KEY,**  **ENAME VARCHAR2(10),**  **JOB VARCHAR2(9),**  **MANAGER NUMBER(4),**  **HIREDATE DATE,**  **SALARY NUMBER(7,2),**  **COMMISSION NUMBER(7,2),**  **DNO NUMBER(2) CONSTRAINT FK\_EMPLOYEE\_DNO REFERENCES DEPARTMENT**  **);**  **CREATE TABLE SALGRADE (**  **GRADE NUMBER,**  **LOSAL NUMBER,**  **HISAL NUMBER**  **);**  **/\* 테이블에 값[레코드] INSERT : DEPARTMENT, EMPLOYEE, SALGRADE \*/**  **INSERT INTO DEPARTMENT VALUES (10,'ACCOUNTING','NEW YORK');**  **INSERT INTO DEPARTMENT VALUES (20,'RESEARCH','DALLAS');**  **INSERT INTO DEPARTMENT VALUES (30,'SALES','CHICAGO');**  **INSERT INTO DEPARTMENT VALUES (40,'OPERATIONS','BOSTON');**  **INSERT INTO EMPLOYEE VALUES**  **(7369,'SMITH','CLERK', 7902,to\_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);**  **INSERT INTO EMPLOYEE VALUES**  **(7499,'ALLEN','SALESMAN', 7698,to\_date('20-2-1981', 'dd-mm-yyyy'),1600,300,30);**  **INSERT INTO EMPLOYEE VALUES**  **(7521,'WARD','SALESMAN', 7698,to\_date('22-2-1981', 'dd-mm-yyyy'),1250,500,30);**  **INSERT INTO EMPLOYEE VALUES**  **(7566,'JONES','MANAGER', 7839,to\_date('2-4-1981', 'dd-mm-yyyy'),2975,NULL,20);**  **INSERT INTO EMPLOYEE VALUES**  **(7654,'MARTIN','SALESMAN',7698,to\_date('28-9-1981', 'dd-mm-yyyy'),1250,1400,30);**  **INSERT INTO EMPLOYEE VALUES**  **(7698,'BLAKE','MANAGER', 7839,to\_date('1-5-1981', 'dd-mm-yyyy'),2850,NULL,30);**  **INSERT INTO EMPLOYEE VALUES**  **(7782,'CLARK','MANAGER', 7839,to\_date('9-6-1981', 'dd-mm-yyyy'),2450,NULL,10);**  **INSERT INTO EMPLOYEE VALUES**  **(7788,'SCOTT','ANALYST', 7566,to\_date('13-07-1987', 'dd-mm-yyyy'),3000,NULL,20);**  **INSERT INTO EMPLOYEE VALUES**  **(7839,'KING','PRESIDENT', NULL,to\_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);**  **INSERT INTO EMPLOYEE VALUES**  **(7844,'TURNER','SALESMAN',7698,to\_date('8-9-1981', 'dd-mm-yyyy'),1500,0,30);**  **INSERT INTO EMPLOYEE VALUES**  **(7876,'ADAMS','CLERK', 7788,to\_date('13-07-1987', 'dd-mm-yyyy'),1100,NULL,20);**  **INSERT INTO EMPLOYEE VALUES**  **(7900,'JAMES','CLERK', 7698,to\_date('3-12-1981', 'dd-mm-yyyy'),950,NULL,30);**  **INSERT INTO EMPLOYEE VALUES**  **(7902,'FORD','ANALYST', 7566,to\_date('3-12-1981', 'dd-mm-yyyy'),3000,NULL,20);**  **INSERT INTO EMPLOYEE VALUES**  **(7934,'MILLER','CLERK', 7782,to\_date('23-1-1982', 'dd-mm-yyyy'),1300,NULL,10);**  **INSERT INTO SALGRADE VALUES (1, 700,1200);**  **INSERT INTO SALGRADE VALUES (2,1201,1400);**  **INSERT INTO SALGRADE VALUES (3,1401,2000);**  **INSERT INTO SALGRADE VALUES (4,2001,3000);**  **INSERT INTO SALGRADE VALUES (5,3001,9999);**  **COMMIT;** |
| --- |

3. Select 문 (1)

3. Select 문 (1)

| /\*  **select : 테이블의 값(레코드)을 검색 (출력), READ**    **select 컬럼명**  **from 테이블명[뷰명];**    **select \***  **from employee;**    \*/  **-- 1. employee 테이블의 모든 컬럼을 출력 하기**  **select \***  **from employee;**  **-- 2. 특정 컬럼만 출력 하기**  **select eno, ename, job**  **from employee;**  **-- 3. 특정 컬럼을 여러번 출력 하기 , 컬럼에 연산을 할 수 있다. 컬럼이름이 바뀔 수 있다.**  **select eno, ename, salary, ename, ename, salary , salary \* 12**  **from employee;**  **-- 4. 컬럼 일름을 별칭으로 바꾸고 출력 하기 [컬럼 alias] , as는 생략 가능**  **select eno as 사원번호, ename 사원명 , salary as 월급, salary \* 12 as 연봉**  **from employee;**  **-- 5. employee 테이블의 각 컬럼의 의미**  **select \***  **from employee;**  **select eno as "사원 번호", ename 사원명, job 직책, manager 직속상관,**  **hiredate 입사날짜, salary 월급, commission 보너스, dno 부서번호**  **from employee;**  **-- 6. 테이블 구조를 확인 하기 , desc 테이블명 , 실제 컬럼명, null 허용여부, 자료형(number : 정수, 실수 , char, varchar2 : 문자열)**  **-- insert 시 컬럼의 자료형, NULL 허용 여부를 확인후 값을 넣음.**  **desc employee;**  **select \***  **from employee;**  **-- 7. 연봉을 계산 : 주의 : NULL과의 연산은 결과가 NULL**  **select eno 사원번호, ename 사원명, salary 월급, commission 보너스**  **from employee;**  **-- 연봉 = 월급 \* 12 + 보너스**  **-- null 컬럼과 연산 을 하면 모두 null**  **-- null 처리가 안됨**  **select eno 사원번호, ename 사원명, salary 월급, commission 보너스, -- 원본 테이블 출력**  **salary \* 12 + commission 연봉**  **from employee;**  **-- nvl ( 컬럼명, 0 ) : null을 처리하는 함수 , 컬럼의 값중 null 을 0으로 변환 하는 함수**  **-- commission 컬럼의 nvl 함수를 적용하여 null 을 0으로 처리후 출력**    **select eno 사원번호, ename 사원명, salary 월급, commission 보너스, -- 원본 테이블 출력**  **nvl(commission,0) 널처리됨 , salary \* 12 + nvl (commission, 0) 연봉**  **from employee;**  **– 8. 컬럼과 컬럼을 연산 할 수 있다. : + , -, \* , / <== number (정수, 실수) 타입만 연산가능**  **-- number : 정수**  **-- number(7,2) : 실수 , 전체 7자리 , 소숫점이하 2자리**  **select \* from employee;**  **desc employee;**  **select ename, salary , salary + salary as "+연산" , salary \* salary as "\*연산", salary \* 0.1 as "월급의10%"**  **from employee;**  **-- 9. 중복된 값을 처리후 출력 : distinct 컬럼명 : 컬럼명의 중복을 제거후 출력**  **select \* from employee;**  **-- 우리회사의 부서만 출력**  **select distinct dno**  **from employee;**  **-- 우리회사의 직책(job)만 중복 제거후 출력 ( CLERK : 사무원, SALESMAN : 영업사원, MANAGER : 관리자, ANALYST : 분석가, PRESIDENT : 사장 )**  **-- 대소문자 구분 하지 않음 : 쿼리 ( select, create, from, group by, order by ), 컬럼명, 테이블명**  **-- 테이블 안의 레코드의 값은 대소문자를 구분함.**    **select distinct job**  **from employee;**  **select \* from employee;**  **-- 10. JOB (직책) 이 MANAGER (관리자) 인 사용자 정보만 출력**  **select ename 사원명, job 직책**  **from employee**  **where job = 'MANAGER' ;**  **-- 11. 부서 테이블 확인 : department**  **select \* from department ;**  **select dno 부서번호 , dname 부서명, loc 부서위치**  **from department;**  **-- salgrade : 연봉의 등급를 적용하는 테이블 <== JOIN**  **select \* from salgrade;**  **select grade 연봉의등급, losal 제일낮은연봉, hisal 높은연봉**  **from salgrade;**  **-- SELECT 문의 전체 내용 , SQL : 구조화된 질의 언어**  **/\***  **SELECT 컬럼명**  **FROM 테이블명[뷰명]**  **WHERE 조건**  **GROUP BY 컬럼명 <=== 컬럼에 동일한 값을 그룹핑해서 처리함.**  **HAVING 조건 <== GROUP BY 결과에 대한 조건**  **ODER BY 컬럼명 ASC[DESC] <=== 컬럼을 정렬해서 출력 , ASC : 내림차순정렬, DESC : 오름차순 정렬**  **\*/**  **-- where 조건 사용하기 : 비교 연산자 (NUMBER 타입의 컬럼에서 사용) : = : 같다, > , < , >=, <=**  **-- 같지 않다 : <> , != , ^=**  **-- or -- in ( 값, 값, 값 )**  **-- and**  **-- between**  **select \* from employee;**  **select \***  **from employee**  **where salary > 1500;**  **select \***  **from employee**  **where salary = 1500;**  **select \***  **from employee**  **where salary <> 1500;**  **-- or 연산자**  **select \***  **from employee**  **where salary = 1500 or salary = 800 or salary = 1600;**  **-- in 키워드 : in ( 값, 값, 값)**  **select \***  **from employee**  **where salary in (1500, 800, 1600) ;**  **-- and 연산자 사용 : 부서번호가 10번인 사원의 연봉이 1500 이상인 사용자를 출력**  **select \***  **from employee**  **where dno = 10 and salary > 1500;**  **-- 직책이 manager 이면서 월급이 2000이상인 사원의 사원번호, 사원명, 월급을 출력**  **select eno 사원번호, ename 사원명 , salary 월급, job 직책**  **from employee**  **where job = 'MANAGER' and salary >= 2000 ;**  **-- 비교 연산자를 date, varchar, char : 81년도 입사한 사원을 모두 출력**  **select \***  **from employee**  **where hiredate > '81/01/01' and hiredate < '81/02/28' ;**  **-- 문자열 : char, varchar, nvarchar, nchar : 문자열 , ' '**  **--**  **select \***  **from employee**  **where ename < 'D' ;**  **-- like : 문자열에서 특정 값만 검색 : % : 모든것이 와도 상관없다, \_ : 한글자가 어떤것이 와도 상관없다.**  **-- 컬럼의 특정 값을 검색시 사용됨.**  **select \* from employee**  **where ename like '%LA%' ;**  **select \* from employee**  **where ename like '\_CO%';**  **-- NULL 검색 : is null , is not null**  **select \* from employee;**  **-- commission 컬럼의 값이 null 인 사용자만 출력 : 보너스가 없는 사원**  **select \* from employee**  **where commission is null ;**  **-- commission 컬럼의 값이 null 인 사용자만 출력 : 보너스가 있는 사원**  **select \* from employee**  **where commission is not null ;**  **-- count () : 레코드의 갯수를 출력 : 페이징 처리. <==**  **-- null 은 카운트 하지 않느다.**  **select count(\*) 총레코드수**  **from employee ;**  **select count(ename)**  **from employee;**  **select count(commission) -- 레코드 수 : null 컬럼은 출력 하지 않는다.**  **from employee;**  **select commission 수능점수 -- null : 비어있는 상태, 0 점**  **from employee;**  **desc employee;**  **-- 테이블의 전체 레코드를 정확하게 출력: not null 컬럼을 count(컬럼명), count(\*)**  **-- 특정 컬럼을 정렬해서 출력 : order by 컬럼명 ASC[DESC} , select 절에서 제일 마지막에 와야 함.**  **-- ASC : 오름차순 정렬 : A =====> Z , 가 ===> 하 , 1==> 9 생략 가능**  **-- DESC : 내림 차순 정렬 : Z=====> A, 하 ===> 가, 9 ===> 1**    **-- 월급(salary) 컬럼을 오름차순 정렬**  **select \***  **from employee**  **order by salary asc ; -- asc는 생략 할 수 있다.**    **-- 월급(salary) 컬럼을 내림차순 정렬**  **select \***  **from employee**  **order by salary desc ;**    **-- 이름(ename) 컬럼을 오름 차순 정렬**  **select \***  **from employee**  **order by ename ;**    **-- 직책 (job) 컬럼을 내림 차순 정렬**  **select \***  **from employee**  **order by job desc;**  **-- 정렬할 컬럼을 where 와 같이 사용할때 order by는 where 절 다음에 처리**  **select \***  **from employee**  **where salary > 1500**  **order by salary desc ;**  **-- order by에서 여러 컬럼을 정렬 할때, 앞의 컬럼을 모두 정렬하고, 같은값이 존재할때 같은 값에 대해서 뒤의 컬럼을 정렬**  **-- 답변형 계시판 만들때 사용됨....**  **select dno , ename**  **from employee**  **order by dno asc, ename desc ;**  **-- job : 오름 차순 정렬, dno : job 의 동일한 값에 대해서 dno를 오름차순,**  **-- ename : dno에서 동일한 값에 대해서 ename을 오름차순정렬**  **select job , dno, ename**  **from employee**  **order by job asc, dno asc, ename asc ;**  **select dno , job , ename**  **from employee**  **order by dno desc, job asc, ename desc ;** |
| --- |

4. Select 문 (2) - 그룹 함수

4. Select 문 (2) - 그룹 함수

| /\* 그룹 함수 : group by, having    select 컬럼명  from 테이블[뷰]  where 조건  group by 컬럼명[동일한값을그룹핑]  having 조건 [group by 한 결과 값의 조건]  order by 컬럼명 asc[desc]  \*/  /\* 집계 함수 : number 타입의 컬럼에 적용 <=== NULL 을 자동으로 처리됨...  SUM - 컬럼의 모든 값의 합  AVG - 컬럼의 모든 값의 합  MAX - 컬럼의 최대값  MIN - 컬럼의 최소값  COUNT (\*) - 그룹핑 된 레코드 수 출력 .    주의 : 단일 행으로 출력이 되기 때문에 다른 컬럼과 사용시 오류.  단 . group by 절에 grouping 된 컬럼은 출력 가능    \*/  desc employee;  -- 컬럼에 함수 사용 :  -- 월급 (salary)  select SUM(salary) 합계 , Round (AVG(salary), 2) 평균, MAX(salary) 최대값, Min (salary ) 최소값 , count(\*) 계산된레코드수  from employee;  -- 집계 함수는 NULL을 자동으로 처리함.  select commission  from employee;  -- 보너스 (commission) <== null은 자동으로 처리됨  select SUM(commission) 합계 , Round (AVG(commission), 2) 평균, MAX(commission) 최대값,  Min (commission ) 최소값 , count(\*) 계산된레코드수  from employee;  -- 부서별로 월급의 합계, 평균, 최대월급 , 최소 월급, 그룹핑된 수를 출력  select dno 부서번호, SUM(salary) 합계, Round ( AVG(salary) ) 평균,  MAX(salary) 최대값, MIN(salary) 최소값, COUNT(\*) 그룹핑된수  from employee  group by dno ; -- DNO 컬럼의 동일한 값을 그룹핑해서 집계함수를 적용함.  select salary, dno  from employee  order by dno asc;  -- 직책(job) 별로 월급의 합계, 평균, 최대값, 최소값, 그룹핑된수를 출력 해 보세요.  select ename, job, salary  from employee  order by job ;  select job 직책 , count(\*) 그룹핑된갯수 , SUM (salary ) 합계,  round(AVG(salary),2) 평균 , MAX(salary) 최대값, MIN(salary) 최소값  from employee  group by job; -- job 컬럼의 동일한 값을 그룹핑  select count(\*) 전체레코드수 from employee;  /\*  group by 절에서 where [조건] vs having [조건]  where [조건] : group by 하기전에 조건을 가져옴. , 별칭이름을 사용할 수 없다.  having [조건] : group by 해서 나온 결과에 대한 조건 , 별칭이름을 사용할 수 없다.    oder by 컬럼명 asc[desc] : 별칭이름을 사용가능  \*/  -- 직급별로 월급의 합계, 평균, 최대값, 최소값, 그룹핑된수를 출력하되, 부서번호 20은 제외하고, 부서별로 평균이 2000이상인 내용만 출력  -- 평균을 desc 출력  select \* from employee  order by dno asc;  select job 직급, count(job) , SUM(salary) 월의합계, Round (AVG(salary)) 평균,  MAX(salary) 최대값, MIN(salary) 최소값  from employee  where dno != 20 -- group by 전의 조건  group by job  having Round (AVG(salary)) >= 2000  order by 평균 desc;  -- group by 절에서 컬럼이 2개 이상 , 두 컬럼에 걸쳐서 동일한 내용을 그룹핑함.  -- job , dno  select job , dno from employee  order by job asc ;  select dno, job, sum (salary) 합한값, count(\*)  from employee  group by job, dno  -- 각 직책(job) 에 대해서 월급의 합계, 평균, 최대값, 최소값, 카운트수를 출력하되 입사일이 81년도 입사한 사원들만 적용  -- 월급의 평균이 1500만원 이상인 것만 출력 하되 평균을 내림차순 정렬  select job 직책, count(\*) , sum(salary) 합계, round(avg(salary)) 평균, max(salary) 최대값,  min(salary) 최소값  from employee  where hiredate like '81%'  group by job  having round(avg(salary)) >= 1500  order by 평균 desc;  /\* group by 절에서 사용되는 키워드  rollup : grouy by 커럼 <== 결과 마지막 라인에 전체 결과도 한번더 출력  cube : 그룹핑 마지막 라인에 결과 출력, 마지막 라인에 전체 결과도 세부적으로 출력  \*/  -- 1. rollup , cube 키워드를 사용하지 않는 경우 : 그룹핑한 결과만 출력  select dno, job , count(\*), SUM(salary), round(avg(salary),2) , max(salary), min(salary)  from employee  group by dno, job  order by count(\*) desc ;    select dno , job  from employee  order by job asc  -- 2. rollup 키워드를 사용하는 경우 : 그룹핑한 결과 출력, 전체 내용도 출력  select dno, job , count(\*), SUM(salary), round(avg(salary),2) , max(salary), min(salary)  from employee  group by rollup ( dno, job )  order by count(\*) desc ;  -- 3. cube 키워드를 사용하는 경우 : 그룹핑한 결과 출력, 전체 내용도 출력, 세부 사항까지 출력  select dno, job , count(\*), SUM(salary), round(avg(salary),2) , max(salary), min(salary)  from employee  group by cube ( dno, job )  order by count(\*) desc ;  /\*  SubQuery : select 문내의 select 쿼리 [ sub query ] , 여러번의 작업을 한 구문에서 처리  - where 절에서 많이 사용됨  \*/  -- ename [사원이름] 이 SCOTT인 사원의 직책과 같은 사원들을 출력 : 2 번 쿼리가 필요함.  -- 1. SCOTT 사원의 직책을 알아오는 쿼리  select job, ename  from employee  where ename = 'SCOTT'    -- 2. 알아온 직책을 조건으로 해서 이름을 출력  select ename , job  from employee  where job = 'ANALYST' ;  -- SubQuery를 사용해서 하나의 쿼리로 처리함.  select ename, job  from employee  where job = ( select job from employee where ename= 'ALLEN' );  select \* from employee ;  -- SCOTT과 ALLEN 의 직책에 해당되지 않는 사원들을 모두 출력  -- 1. scott의 직책을 출력하는 쿼리  -- 2. allen의 직책을 출력하는 쿼리  -- 3. where job not in ('scott의직책', 'allen 직책')  -- 1. scott의 직책을 출력하는 쿼리  select job from employee where ename = 'SCOTT'; -- ANALYST  -- 2. allen의 직책을 출력하는 쿼리  select job from employee where ename = 'ALLEN'; -- SALESMAN    --3. SCOTT과 ALLEN 의 직책에 해당되지 않는 사원들을 모두 출력  select ename , job  from employee  where job not in ( 'ANALYST', 'SALESMAN');    --- SubQuery로 한 라인으로 출력    select ename , job  from employee  where job not in (  (select job from employee where ename = 'SCOTT'),  (select job from employee where ename = 'ALLEN')  );    select ename , job  from employee  where job not in (  select job from employee  where ename = 'SCOTT' or ename = 'ALLEN'  );    select ename , job  from employee  where job not in (  select job from employee  where ename in ( 'SCOTT' , 'ALLEN' )  );  -- 단일 값 : =  -- 여러개의 값 : in    -- 'SCOTT' 보다 않은 월급을 받는 사원들 정보, 월급를 출력  select ename 사원, salary 월급  from employee  where salary > ( select salary from employee where ename = 'SCOTT')      select \* from employee  order by salary desc ;    -- 최소 급여를 받는 사원정의 이름, 담당업무, 급여 출력  select ename, job, salary  from employee  where salary = ( select MIN(salary) from employee );  -- subquery를 사용해서 출력  -- 부서별(dno) 로 최소 급여를 받는 사원정보의 이름, 직책, 월급 을 출력 : 힌트 : group by , dno , min , in 키워드  select salary, dno , ename  from employee  order by dno asc , salary desc ;  -- subquery  select ename 이름, job 직책, salary 월급 , dno 부서번호  from employee  where salary in (  -- 각 부서별 최소월급의 select 한 값을 출력  select min(salary)  from employee  group by dno  ); |
| --- |

5. Select 문 (3) - 제약조건, 조인,뷰

5. Select 문 (3) - 제약조건, 조인,뷰

| /\*  JOIN : DataBase 에는 많은 테이블이 존재합니다. 모델링을 통해서 테이블이 분리 되어 있다. ( R-DBMS)  employee 테이블과 department 테이블은 하나의 테이블인데 모델링(1,2,3정규화), 을 통해서 테이블을 분리  모텔링 : 중복 제거, 성능향상    제약 조건 : 테이블의 컬럼에 들어가는 키 , 데이터의 무결성 (결함없는데이터, 원하는 값만 )  Primary Key :  - 테이블의 컬럼에 1번만 넣을 수 있다.  - 두개 컬럼을 묶어서 PK를 넣을 수 있다.  - 테이블을 생성할때 반드시 PK 이 존재해야 한다.  - Update, Delete 구문에서 PK 컬럼을 where 조건으로 사용함.  - 특정 컬럼에 중복 된 값을 넣지 못하도록 함.  - 반드시 not null 컬럼이어야함. null을 넣을 수 없다.  - index 가 자동으로 생성 된다. 컬럼의 검색을 빠르게 함.  - JOIN 시 ON 에서 많이 사용하는 키 컬럼.  Unique Key :  - 컬럼에 중복된 값을 넣지 못하도록 함.  - null을 넣을 수 있다. 단 1번만 넣을 수 있다. not null, null  - 하나의 테이블에 여러번 Unique Key 를 넣을 수 있다.  - index 가 자동으로 생성됨. JOIN 시 ON에 사용됨  Foreign Key :  - 다른 테이블(부모 테이블) 의 특정 컬럼을 참조 해서 값을 넣도록 함.  - Foreign Key 가 잠조하는 컬럼은 부모테이블의 Primary Key , Unique Key 를 참조함.  NOT NULL  - 컬럼에 NULL을 넣을 수 없도록 하는 제약 조건  CHECK  - 컬럼에 조건을 넣어서 내개 원하는 값만 넣을 수 있도록 함.  - 월 컬럼에 1 ~ 12 까지 넣을 수 있도록  default : 제약 조건은 아니지만 제약 조건처럼 사용됨  - 컬럼에 값을 넣지 않으면 default 로 설정된 값이 등록됨.  \*/  select \* from employee; -- employee 테이블의 dno 컬럼은 참조한다 (Foreign Key )  -- department 테이블의 dno 컬럼을 참조한다.  select \* from department;  -- 테이블 복사 : 원본 테이블의 제약 조건은 복사 되어 오지 않는다. Alter Table 을 사용해서 제약 조건을 부여  create table emp01  as  select \* from employee;  create table dept01  as  select \* from department;  /\*  테이블의 제약 조건을 확인 하는 명령어  user\_constraints : 데이터 사전 <== 테이블의 각종 정보를 알려주는 테이블  \*/  select \* from user\_constraints ;  select \* from user\_constraints  where table\_name in ('EMPLOYEE', 'DEPARTMENT') ;  -- 테이블을 복사하면 테이블의 컬럼에 부여된 제약 조건은 복사 되어 오지 않는다. 컬럼과 값만 복사되어온다.  -- alter table 을 사용해서 제약 조건을 부여해야 함.  -- alter table : 생성된 테이블을 수정  select \* from user\_constraints  where table\_name in ('EMP01', 'DEPT01') ;  -- dept01 테이블에 dno컬럼에 Primary Key 제약 조건을 추가  alter table dept01  add constraint PK\_DEPT01\_DNO primary key (dno);  -- emp01 테이블에 eno 컬럼에 Primary Key 제약 조건 추가  alter table emp01  add constraint PK\_EMP01\_ENO primary key (eno);  -- emp01 테이블의 dno 컬럼에 Foreign Key 부여 , 참조(references)할 테이블의 컬럼은 Dept01 테이블의 dno 컬럼을 참조  alter table emp01  add constraint FK\_EMP01\_DNO foreign key (dno) references dept01(dno);  -- Primary Key 컬럼을 확인 :  desc emp01 ;  select \* from emp01 ;  -- 값을 넣을때 컬럼에 부여된 제약 조건을 잘 확인 하고 값을 insert  -- eno : Primary Key 가 등록, 중복된 값을 넣으면 안됨  -- dno : Foreign Key 등록 , Dept01 테이블의 dno 컬럼에 존재하는 값만 넣어야 한다.  insert into emp01 ( eno, ename, job, manager, hiredate, salary, commission, dno )  values ( 7977, 'PJW', 'CLERK', 7782 , '23/12/11', 1500 , null , 40 );  commit ; -- DML ( insert, update, delete ) 에서 DB에 영구히 저장되도록 함. <오라클>  select \* from emp01;  select \* from dept01;  -- dept 테이블에 값 넣기 : dno 컬럼 : Primary key 가 있음.  desc dept01 ;  insert into dept01 ( dno, dname, loc)  values ( 50 , 'HR', 'SEOUL') ;  commit ; -- DML ( insert, update, delete) 에서 반드시 적용.  select \* from dept01;  select \* from emp01 ;  -- JOIN : 여러 테이블의 컬럼을 출력 할때 JOIN을 사용해서 하나의 테이블 처럼 출력함.  -- 두 테이블의 공통 키 컬럼을 확인,  -- emp01, dept01 테이블의 공통 키 컬럼은 dno 이다.  -- EQUI JOIN : 오라클에서만 작동하는 JOIN 구문  -- ANSI JOIN : 모든 DBMS에서 공통으로 사용되는 JOIN 구문    -- EQUI JOIN 구문으로 두 테이블 조인  -- from 절에서 JOIN 할 테이블을 명시 (,)  -- 테이블 이름은 별칭이름으로 둠.  -- where 절에서 두 테이블의 공통 키 컬럼을 명시  -- and 절에서 조건을 처리  -- 공통 키 컬럼을 출력시 반드시 테이블명.컴럼명    select e.eno, e.ename, e.job,d.dno , d.dname, d.loc  from emp01 e , dept01 d  where e.dno = d.dno  select eno, ename, job, d.dno , dname, loc  from emp01 e , dept01 d  where e.dno = d.dno  and d.dno = 10 ;  -- ANSI JOIN : 모든 DBMS에서 공통으로 사용되는 JOIN 구문  -- INNER JOIN : 두 테이블에서 키컬럼에 공통 되는 부분만 출력 <=== 80 %  -- OUTER JOIN :  -- LEFT OUTER JOIN  -- RIRGHT OUTER JOIN  -- FULL OUTER JOIN  -- SELF JOIN  -- CLOSS JOIN  -- INNER JOIN : ANSI JOIN  -- from 절에 JOIN 테이블 이름을 명시 ,  -- INNER 키는 생략 될 수 있다. <=== 80 % 이상  -- on 절에 두 테이블의 공통 키 컬럼을 명시 , 두 테이블의 공통인 것만 출력  -- where 절에서 조건 처리  -- 테이블 이름을 alias (별칭) 시키지 않는 경우  select eno, ename, salary,dept01.dno, dname, loc  from emp01 INNER JOIN dept01  on emp01.dno = dept01.dno  -- 테이블 이름을 alias (별칭) 사용한 경우  select eno, ename, salary, d.dno, dname, loc  from emp01 e JOIN dept01 d  on e.dno = d.dno  where e.dno = 20  order by ename desc ;  -- INNER JOIN : ANSI SQL : 모든 DBMS에서 공통으로 사용되는 SQL 퀄리 ( Oracle, MySQL, MSSQL, DB2, .....)  -- ON 절에 두 테이블의 공통분모만 출력  select \*  from emp01 e JOIN dept01 d  on e.dno = d.dno  -- OUTER JOIN :  -- LEFT OUTER JOIN : 왼쪽 테이블의 모든 내용을 출력함.  -- RIGHT OUTER JOIN : 오른쪽 테이블의 모든 내용을 출력함.  -- FULL OUTER JOIN : 왼쪽, 오른쪽 테이블의 모든 내용을 출력함.  select \* from dept01;  insert into dept01 (dno , dname, loc)  values ( 60 , 'MANAGE', 'PUSAN');  commit ;  -- 가끔 사용 함.  -- RIGHT OUTER JOIN : 두 테이블의 공통 부분과 오른쪽 (dept01) 테이블의 모든 값을 출력  select \*  from emp01 e RIGHT OUTER JOIN dept01 d  on e.dno = d.dno  -- FULL OUTER JOIN : 두 테이블(왼쪽, 오른쪽)의 모든 내용을 출력  select \*  from emp01 e FULL OUTER JOIN dept01 d  on e.dno = d.dno ;  -- SELF JOIN : 자신의 테이블을 JOIN, 자시의 테이블을 별칭이름으로 가상으로 생성해서 JOIN  -- 조직도 출력, 직급 상사를 바로 출력 할때 사용됨    select e.eno 사원번호, e.ename 사원이름, e.manager 직속상관번호, m.eno 직속상관사번, m.ename 직속상관명  from emp01 e JOIN emp01 m  on e.manager = m.eno ;  -- SELF JOIN으로 사원이름에대한 직속 상관이 누구인지 출력  -- 직속상관이 없는 사원을 출력 : LEFT OUTER JOIN  select e.eno 사원번호, e.ename 사원이름, e.manager 직속상관번호, m.eno 직속상관사번, m.ename 직속상관명  from emp01 e LEFT OUTER JOIN emp01 m  on e.manager = m.eno ;  -- 사원번호는 있지만 어떤 사원의 상관이 아닌 사원도 모두 출력 해라 : RIGHT OUTER JOIN  select e.eno 사원번호, e.ename 사원이름, e.manager 직속상관번호, m.eno 직속상관사번, m.ename 직속상관명  from emp01 e RIGHT OUTER JOIN emp01 m  on e.manager = m.eno ;  -- 직속 상관이 없는것 (왼쪽), 사원번호는 가지지만 어떤사원의 직속 상관이 아닌 사원(오른쪽) : FULL OUTER JOIN  select e.eno 사원번호, e.ename 사원이름, e.manager 직속상관번호, m.eno 직속상관사번, m.ename 직속상관명  from emp01 e FULL OUTER JOIN emp01 m  on e.manager = m.eno ;  select eno , ename , manager, eno, ename  from employee;  --1. 사원이름 'SCOTT' 의 부서명(dname), 부서위치(loc) <=== ename : 'SCOTT'은 emp01, dept01 : dname, loc  -- ANSI JOIN  select \* from emp01 ;  select \* from dept01;  -- emp01 , dept01 을 JOIN 해서 출력  select ename, dname,loc ,e.dno  from emp01 e JOIN dept01 d  on e.dno = d.dno  where ename = 'SCOTT';  --2. 월급이 2000만원 이상인 사원의 이름, 월급, 부서명, 부서위치, 을 출력 : ANSI JOIN  select ename, salary, dname, loc , e.dno  from emp01 e JOIN dept01 d  on e.dno = d.dno  where salary >= 2000  order by d.dno desc;  -- ANSI JOIN  --3 . 직책(job) 이 'MANAGER' 인 사원이름(ename), 부서번호(dno), 부서명(dname), 부서위치(loc) 출력하되 사원이름 내림차순 정렬  select job 직책, ename 사원이름, e.dno 부서번호, dname 부서명, loc 부서위치  from emp01 e JOIN dept01 d  on e.dno = d.dno  where job = 'MANAGER';  -- VIEW ( 뷰 ) : 가상의 테이블 , 값은 가지지 않고, 코드 만 가진다.  -- 1. 실제 테이블의 특정 컬럼만 출력 할때 <== 보안  -- 2. 복잡한 쿼리를 한번 만들어서 실행 <== 복잡한 JOIN 쿼리를 단순화 할 수 있다.  -- view 를 실습하기위한 샘플테이블 : emp02  drop table EMP02 cascade constraints ;  create table EMP02  as  select eno, ename , salary , commission, job, hiredate , dno  from employee  where salary > 1500;  select \* from emp02;  create table dept02  as  select \* from department;  -- 기존의 원본테이블 : employee, department ===> emp01, dept01  -- 필드명, 값 만 복사됨  -- 컬럼에 부여된 제약 조건을 복사되지 않음. <== Alter Table 을 사용해서 제약 조건 추가    select \* from user\_constraints where table\_name in ('EMPLOYEE', 'DEPARTMENT');  select \* from user\_constraints where table\_name in ('EMP02', 'DEPT02');  --1. DEPT02 테이블의 dno (Primary Key)  --2. EMP02 테이블의 eno (Primary Key), dno (FK) ==> dept02(dno)  alter table DEPT02  add constraints PK\_DEPT02\_DNO primary key (dno) ;  alter table EMP02  add constraints PK\_EMP02\_ENO primary key (eno);  alter table EMP02  add constraints FK\_EMP02\_DNO foreign key (dno) references dept02(dno);  -- VIEW 생성 1. 실제 테이블의 중요 정보를 감추고 출력 , 보안을 향상할 수 있다.  -- as 다음에 select 코드를 가지고 있음.  select \* from emp02 ; -- 실제 테이블 : emp02  -- 뷰 생성 : 실제 테이블의 값을 가지는 것이 아니라 실행 코드만 가진다.  create view v\_emp02  as  select ename, job, dno  from emp02 ;  -- 뷰 실행 :  select 컴럼명 from 테이블명[뷰명]  select \* from v\_emp02;  -- 데이터 사전 : user\_테이블 <== 시스템의 정보을 저장되어있는 테이블  select \* from user\_views;  -- 2. 사용의 편의성 , 복잡한 구문을 view를 생성해서 저장시켜 두고 실행, <== JOIN  -- 두 테이블을 조인해서 월급이 2500 이상인 사원 정보를 출력  create view v\_join  as  select ename, job, salary, dname, loc  from emp02 e  join dept02 d  on e.dno = d.dno  where salary >= 2500 ;  select \* from v\_join ;  -- employee, department 테이블의 부서별로 최소 월급을 받는 사원이름, 사원의 직책, 부서명, 부서위치 를 출력되 최소월급이 900이상  -- 사원 정보를 출력 , 부서번호 20번은 제외하고  -- 뷰안에 쿼리를 저장하고 뷰를 실행해서 출력 <== JOIN, group by, where, having , subquery    -- 부서별로 최소 월급을 받는 사원 정보를 출력하는 view , 20 부서는 제외됨  create view v\_join2  as  select ename 사원이름, job 직책, dname 부서명, loc 부서위치, d.dno 부서번호  from employee e  join department d  on e.dno = d.dno  where salary in (  -- 부서별로 최소 월급  select min(salary) from employee  where dno <> 20  group by dno  having min(salary) > 900  )  -- 뷰 실행  select \* from v\_join2;  -- view 에 정렬 컬럼 까지 적용됨  create view v\_join3  as  select ename 사원이름, job 직책, dname 부서명, loc 부서위치, d.dno 부서번호  from employee e  join department d  on e.dno = d.dno  where salary in (  -- 부서별로 최소 월급  select min(salary) from employee  where dno <> 20  group by dno  having min(salary) > 900  )  order by ename desc  select \* from v\_join3  -- view 에 as 블락에 select 문이 옴  -- 실제 값을 가지고 있지 않음. select 코드만 들어 있음.  -- 실제 테이블의 컬럼에 잘 매칭 될경우  -- insert, update, delete를 view 통해서 할 수 있다. 단 실제 테이블에 각 컬럼의 제약 조건에 맞을때 가능  select \* from emp02 ;  -- view 생성  drop view v\_test01 ;  create view v\_test01  as  select eno, ename , dno  from emp02;  -- view 실행  select \* from v\_test01;  --view 에 값을 넣을 수 있다 ==> 실제 테이블에 값이 들어감.  insert into v\_test01 (eno, ename, dno)  values ( 8080, 'HONG', 30);  commit;  -- view 를 사용해서 값을 수정 : update <== where 조건을 반드시 사용해야하고 컬럼명은 Primary Key 가 들어가 컬럼  update v\_test01  set ename = 'KIM'  where eno = 8080 ;  commit ;  -- view 를 사용해서 값을 삭제 : delete <== where 조건을 반드시 사용해야하고 컬럼명은 Primary Key 가 들어가 컬럼  delete v\_test01  where eno = 8080 ;  commit ;  select \* from emp02  -- v\_test02 : insert (안됨) , update (가능), delete( 가능) ==> 실제 테이블의 제약 조건에 잘 맞아야 한다.  create view v\_test02  as  select eno, ename, salary  from emp02 ;  insert into v\_test02 (eno, ename , salary)  values ( 9090 , 'SIM', 3500);  -- 테이블의 특정 컬럼에 not null 제약 조건 추가  alter table emp02  modify dno NOT NULL ;  delete v\_test02  where eno = 9090;  commit; |
| --- |

6. DDL 과 제약 조건, Alter Table

6. DDL 과 제약 조건 , Alter Table

| /\* DDL : create (생성), alter (수정) , drop (삭제) <== 테이블, 뷰, 함수, 시퀀스, 트리거, 저장프로시져 , 스카마를 생성 , 수정, 삭제  자료형 : 데이터를 저장하는 타입  - 숫자 : number (3) : 정수 3자리  number ( 7, 2 ) : 소수 , 전체 7자리 , 소숫점이하 2자리 까지  - 문자 : char (n) ; 영문 1자 (1byte) , 한글 1자 (3byte)  ` - 성능이 빠르다. 하드 공간 낭비가 발생할 수 있다.  - 주민번호(13) , 자릿수 가 지정된 곳에 사용됨  varchar2 (n) : 영문 1자 (1byte) , 한글 1자 (3byte)  - 가변공간으로 적용됨 , 성능은 char 보다 느릴수 있다. 하드공간 낭비를 시키지 않는다.  - 자릿수를 알수 없는 문자열 때 사용됨  nchar(n) : nchar(10) : 한글 10자  nvarchar2(n) : nchar(10) : 한글 10자  - 날짜 : date : BC 4712년 1월 1일 ~~~ 9999년 12월 31일까지저장, 년, 월, 일 시 분, 초 까지만 저장 가능  timestamp : insert (값이 들어오는 시스템의 시간) : 년, 월, 일, 시, 분,초 밀리센컨드 까지 저장  - LOB 데이터 타입 : 대량의 값을 저장, 바이너리 파일  CLOB : 문자를 대량으로 넣을수 있다. <== 글 내용  BLOB : mp3, jpeg, hwp 이진 데이터 파일  BFile : 대용량 파일 저장  \*/  --  drop table test10 ;  create table test10 (  id number(4) not null primary key , -- 중복된 넣을 수 없다. 정수 4자  n1 char(10) , -- 영문 10자, 한글 3자  n2 nchar(10) , -- 영문 10자 , 한글 10자  n3 varchar2(10) , -- 영문 10자, 한글 3자  n4 nvarchar2(10) -- 영문 10자 , 한글 10자  );  select \* from test10 ;  commit;  insert into test10 ( id , n1 , n2, n3, n4)  values ( 1111, 'abcdefghij', '가나다라마바사카차타' , 'abcdefghij', '가나다라마바사카차타' ) ;  insert into test10 ( id , n1 )  values ( 1112, '가나다') ;  /\*  제약 조건 : 테이블의 컬럼에 들어가는 키 , 데이터의 무결성 (결함없는데이터, 원하는 값만 )  Primary Key :  - 테이블의 컬럼에 1번만 넣을 수 있다.  - 두개 컬럼을 묶어서 PK를 넣을 수 있다.  - 테이블을 생성할때 반드시 PK 이 존재해야 한다.  - Update, Delete 구문에서 PK 컬럼을 where 조건으로 사용함.  - 특정 컬럼에 중복 된 값을 넣지 못하도록 함.  - 반드시 not null 컬럼이어야함. null을 넣을 수 없다.  - index 가 자동으로 생성 된다. 컬럼의 검색을 빠르게 함.  - JOIN 시 ON 에서 많이 사용하는 키 컬럼.  Unique Key :  - 컬럼에 중복된 값을 넣지 못하도록 함.  - null을 넣을 수 있다. 단 1번만 넣을 수 있다. not null, null  - 하나의 테이블에 여러번 Unique Key 를 넣을 수 있다.  - index 가 자동으로 생성됨. JOIN 시 ON에 사용됨  Foreign Key :  - 다른 테이블(부모 테이블) 의 특정 컬럼을 참조 해서 값을 넣도록 함.  - Foreign Key 가 잠조하는 컬럼은 부모테이블의 Primary Key , Unique Key 를 참조함.  NOT NULL  - 컬럼에 NULL을 넣을 수 없도록 하는 제약 조건  CHECK  - 컬럼에 조건을 넣어서 내개 원하는 값만 넣을 수 있도록 함.  - 월 컬럼에 1 ~ 12 까지 넣을 수 있도록  default : 제약 조건은 아니지만 제약 조건처럼 사용됨  - 컬럼에 값을 넣지 않으면 default 로 설정된 값이 등록됨.    제약 조건을 출력하는 데이터 사전 : user\_constraints  select \* from user\_constraints where table\_name in ('테이블명') ;  \*/  -- 제약 조건 이름을 넣지 않고 테이블 생성 한 경우 : Oracle에서 제약조건이름을 랜덤하게 생성한다.  -- insert 시 오류가 발생할 경우 제약조건이름으로 오류난 컬럼을 찾기 힘들다.  create table member1 (  id varchar2(50) not null primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.  pass varchar2(50) not null,  addr varchar2(100) null ,  jumin char(13) null , -- 자릿수가 지정된 컬럼  phone varchar2(50) ,  age number(3) , -- 정수 3자리  weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리  ) ;  desc member1 ;  insert into member1 (id , pass, addr, jumin, phone, age, weight)  values ( 'abc', '1234', '서울', '123456-789101', '010-1111-1111', 30 , 77.77);  commit ;  select \* from member1;  select \* from user\_constraints where table\_name in ('MEMBER1') ;  -- 테이블 생성시 제약 조건 이름 부여  create table member2 (  id varchar2(50) not null constraint PK\_MEMBER2\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.  pass varchar2(50) constraint NN\_MEMBER2\_PASS not null,  addr varchar2(100) null ,  jumin char(13) null , -- 자릿수가 지정된 컬럼  phone varchar2(50) ,  age number(3) , -- 정수 3자리  weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리  ) ;  select \* from user\_constraints where table\_name in ('MEMBER2') ;  insert into member2 (id , addr, jumin, phone, age, weight)  values ( 'abcd', '서울', '123456-789101', '010-1111-1111', 30 , 77.77);  commit ;  /\* UNIQUE : 중복된 값을 넣을수 없다. null을 넣을 수 있다. 테이블에 여러번 넣을수 있다.  \*/  create table member3 (  id varchar2(50) not null constraint PK\_MEMBER3\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.  pass varchar2(50) constraint NN\_MEMBER3\_PASS not null,  addr varchar2(100) null ,  jumin char(13) null constraint U\_MEMBER3\_JUMIN unique , -- 중복되면 안됨  phone varchar2(50) not null constraint U\_MEMBER3\_PHONE unique , -- 중복되면 안됨  age number(3) , -- 정수 3자리  weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리  ) ;  insert into member3 (id , pass, addr, jumin, phone, age, weight)  values ( 'abc', '1234', '서울', '123456-789102', '010-1111-1112', 30 , 77.77);  commit ;  -- CHECK 제약 조건 : 컬럼에 조건에 맞는 값만 넣을 수 있도록 함.  drop table member4 ;  create table member4 (  id varchar2(50) not null constraint PK\_MEMBER4\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.  pass varchar2(50) constraint NN\_MEMBER4\_PASS not null,  addr varchar2(100) null constraint CK\_MEMBER4\_ADDR check (addr in ('서울', '부산','대구')) ,  jumin char(13) null constraint U\_MEMBER4\_JUMIN unique , -- 중복되면 안됨  phone varchar2(50) not null constraint U\_MEMBER4\_PHONE unique , -- 중복되면 안됨  age number(3) constraint CK\_MEMBER4\_AGE check (age > 0 and age < 200) , -- 정수 3자리  gender char(1) constraint CK\_MEMBER4\_GENDER check ( gender in ('w', 'm')),  weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리  ) ;  insert into member4 (id , pass, addr, jumin, phone, age, weight, gender )  values ( 'abc', '1234', '대구', '123456-789102', '010-1111-1112', 100, 77.77, 'w');  commit ;  -- default : 제약조건이 아니다. 제약조건 이름을 부여 할 수 없다.  -- 값을 넣을때 값이 들어가고 값을 넣지 않을때 default 로 설정된 값이 들어간다.  create table member5 (  id varchar2(50) not null constraint PK\_MEMBER5\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.  pass varchar2(50) constraint NN\_MEMBER5\_PASS not null,  addr varchar2(100) null constraint CK\_MEMBER5\_ADDR check (addr in ('서울', '부산','대구')) ,  jumin char(13) null constraint U\_MEMBER5\_JUMIN unique , -- 중복되면 안됨  phone varchar2(50) not null constraint U\_MEMBER5\_PHONE unique , -- 중복되면 안됨  age number(3) constraint CK\_MEMBER5\_AGE check (age > 0 and age < 200) , -- 정수 3자리  gender char(1) constraint CK\_MEMBER5\_GENDER check ( gender in ('w', 'm')),  weight number (5,2), -- 실수 전체 5자리, 소숫점이하 2자리  hiredate date default sysdate,  addr2 char(10) default '서울' ,  age2 number default 0  ) ;    select \* from member5;  insert into member5 (id , pass, addr, jumin, phone, age, weight, gender )  values ( 'abc', '1234', '대구', '123456-789102', '010-1111-1112', 100, 77.77, 'w');  insert into member5 (id , pass, addr, jumin, phone, age, weight, gender , hiredate, addr2, age2 )  values ( 'abcd', '1234', '대구', '123456-789103', '010-1111-1113', 100, 77.77, 'w', '22/11/11', '광주', 30);  commit ;  -- 날짜 타입에 체크 제약 조건  CREATE TABLE emp55 (  e\_id NUMBER PRIMARY KEY, -- 아이디 컬럼  h\_date DATE CHECK (h\_date BETWEEN TO\_DATE('1981-01-01', 'YYYY-MM-DD') AND TO\_DATE('1981-12-31', 'YYYY-MM-DD'))  );  insert into emp55  values ( 10, '81/11/11');  commit ; |
| --- |

07. DDL DML

**07. DDL DML**

/\*

SQL : 모든 쿼리 구문 , select, create, alter, drop , insert, update, delete, grant, revoke, rollbac, commit

- 구조화된 질의 언어

SQL 종류

- DQL(Data Query Language) : 출력, 검색 , select

- DDL (Date Definition Language) : 스키마 (틀, 구조), 객체 (테이블, 뷰, 함수, 시퀀스, 트리거, 저장프로시져, 인덱스) 생성, 수정,삭제

create (생성), alter (수정), drop(삭제)

- DML ( Data Manupulation Language) : 테이블의 레코드(값) 를 조작하는 언어

insert( 입력 ) , update( 수정 ) , delete ( 삭제 )

-- 트랜잭션이 발생( RAM ) , commit : RAM의 변경된 내용을 DataBase 영구히 저장하도록 함. , rollback : 원래대로 되돌림.

-- 트랜Ъ퓽 insert, update, delete 시작되면 자동으로 트랜잭션이 시작

-- 트랜잭션을 종료 : rollback : 메모리에 변경된 내용을 다시 원래 상태로 되돌림, commit : DB에 영구적으로 적용함.

-- 트랜Ъ퓽 종료 되지 않으면 LOCK 걸려서 트랜Ъ 트랜잭션 종료될때 까지 LOCK 걸려 있다.

-- 오라클 : DML 문을 시작하면 자동으로 트랜잭션이 시작됨 . 종료 (rollback, commit) <== 명시

-- MySQL, MSSQL : DML 문을 시작하면 자동으로 트랜잭션이 시작됨 . commit 자동으로 처리됨

-- 명시적으로 트랜잭션을 시작 할 수 있고, 명시적으로 끝낼 수 있다.

- DCL ( Data Control Language) : 계정(Account) 에게 객체(Resource) 에 권한을 부여함. grant(부여), revoke (제거)

- TCL ( Transaction Control Language) : 트랜잭션을 관리 하는 언어, <== DML 문에서 사용됨

commit (DB에 영구히 저장), rollback ( 원래되로 되돌림), savepoint ( 트랜잭션 내에서 임시 저장시점 생성 )

begin transaction : 명시적으로 트랜잭션 시작

\*/

/\*

트랜잭션 : 작업(일) 을 처리하는 최소 단위 : ALL or NOTHING : 되면 전부 되게 하거나 안되면 전부 안되게 함.

- 트랜잭션 log 에 기록 되어 있다. <== 오류난 시점까지 복원 , 백업 시점 시점이 아니라 오류난 시점까지 복원가능

- 트랜잭션의 4가지 특징

- 원자성 (Atomicity) : 일을 처리하는 최소 단위

- 일관성 (Consistency) : 트랜잭션에서 처리된 결과는 일관성을 가진다. ALL or NOTHING

- 독립성 (lsolation) : 하나의 트랜잭션은 다른 트랜잭션과 격리되어 있다. , LOCK,

- 직속성 ( Durability) : commit , DB에 영구적으로 저장됨

\*/

-- 테이블 복사

create table account10

as

select eno as no, ename as name , salary as account

from employee ;

-- 은행의 통장 테이블 : no : 계좌번호, name : 통장주인 , account : 돈 (입금액), 억

select \* from account10;

-- SMITH 에서 Miller에게 10억을 입금 : 2개의 update 구문을 하나의 transaction 으로 처리 : ALL or NOTHING

-- SMITH Account - 10억 : update account10 set account = account - 10 where no= 7369

-- Miller Account + 10억 : update account10 set account = account + 10 where no = 7934

begin transaction ; -- 명시적으로 트랜잭션을 시작

update account10

set account = account - 10

where no = 7369;

update account10

set account = account + 10

where no = 7934;

select \* from account10;

-- 트랜잭션 종료

rollback;

commit ;

desc account10 ;

/\*

DML : insert(값을 입력) , update (수정) , delete (삭제)

<== 트랜잭션 자동 시작,

<== 트랜잭션 종료 ( rollback, commit) 는 명시 (오라클)

\*/

-- 테이블 복사

create table dept03

as

select \* from department ;

select \* from dept03 ;

-- dept03 : primary key 제약 조건 추가

alter table dept03

add constraint PK\_DEPT03\_DNO primary key (dno) ;

-- INSERT : 주의 사항 , 각 컬럼에 자료형 : number , 문자, 날짜 , 컬럼에 적용된 제약 조건을 확인

desc dept03 ;

select \* from user\_constraints where table\_name in ('DEPT03') ;

-- insert into 테이블명 (컬럼명, 컬럼명,컬럼명) values ( 값, 값, 값 ) ;

insert into dept03 ( dno, dname, loc )

values ( 50, 'HR', 'SEOUL') ;

rollback ;

commit ;

select \* from dept03 ;

-- insert 시 컬럼 이름을 명시 하지 않는 경우 모든 컬럼에 값을 넣어야함.

insert into dept03

values ( 60, '인사부', '부산');

commit ;

-- 컬럼 생략 할때는 모든 컬럼에 값이 순서에 맞게 입력

insert into dept03

values ( 70, '인사부');

-- 컬럼을 명시할때 순서를 바꿀 수 있고 , 특정컬럼은 값을 넣지 않아도 됨.

insert into dept03 ( dname, dno)

values ( '영업부', 80) ;

commit ;

--UPDATE 문 : 입력된 값을 수정시 사용 , 반드시 where 조건을 사용, where 조건에 사용되는 컬럼은 Primary Key 컬럼이어야 함.

/\*

update 테이블명

set 컬럼명 = 바꿀값 , 컬럼명=바꿀값

where 조건

\*/

update dept03

set loc = '대구'

where dno = 80 ;

rollback ;

commit ;

select \* from dept03 ;

--

update dept03

set dname = '관리부', loc = '광주'

where dno = 40 ;

--

insert into dept03

values ( 90, '영업부', '대구') ;

insert into dept03

values ( 91, '영업부', '광주') ;

-- update 에서 반드시 where 조건 , 조건을 처리하는 컬럼은 Primary key, Unique 키 컬럼을 정의

update dept03

set dname = '인쇄부'

where loc = '광주' ; -- 중복된 값

-- update 시 primary key 컬럼을 조건으로 처리해서 원하는 값만 수정

update dept03

set dname = '인쇄부'

where dno in ( 91 , 80) ;

-- Delete : 레코드를 삭제 할때 , where <조건> , <조건 : 중복되지 않는 컬럼>

/\*

delete 테이블명

where 조건

\*/

select \* from dept03;

-- delete 문에서 조건을 사용하지 않는 경우 모든 레코드가 삭제됨 ,

delete dept03

rollback ;

commit ;

-- delete , where 조건 사용 , <==

delete dept03

where dno = 91 ;

-- 모든 레코드를 삭제 :

-- delete : 레코드 하나 하나를 삭제함. 시간이 오래 걸림. << 로우레벨 포멧 >>

-- truncate table 테이블명 : 한꺼번에 통채로 모든 레코드를 날림 , <<빠른 포멧>>

-- drop table 테이블명 : 테이블 자체를 삭제

select \* from dept03;

rollback ;

commit ;

-- 모든 레코드 삭제 : 값만 삭제

delete dept03 ;

-- 모든 레코드 삭제 : 값만 삭제 <== DDL 문이기 때문에 Rollback이 안됨. 실행후 바로 커밋됨

-- 빠르게 처리됨

truncate table dept03 ;

-----------------------------------------------------------

create table emp04

as

select \* from employee ;

select \* from emp04 ;

/\* emp04

임의의 값을 추가 : insert

임의의 값을 수정 : update

임의의 값을 삭제 : delete

rollback ;

commit ;

\*/

/\* DDL : create (생성), alter (수정) , drop (삭제) <== 테이블, 뷰, 함수, 시퀀스, 트리거, 저장프로시져 , 스카마를 생성 , 수정, 삭제

자료형 : 데이터를 저장하는 타입

- 숫자 : number (3) : 정수 3자리

number ( 7, 2 ) : 소수 , 전체 7자리 , 소숫점이하 2자리 까지

- 문자 : char (n) ; 영문 1자 (1byte) , 한글 1자 (3byte)

` - 성능이 빠르다. 하드 공간 낭비가 발생할 수 있다.

- 주민번호(13) , 자릿수 가 지정된 곳에 사용됨

varchar2 (n) : 영문 1자 (1byte) , 한글 1자 (3byte)

- 가변공간으로 적용됨 , 성능은 char 보다 느릴수 있다. 하드공간 낭비를 시키지 않는다.

- 자릿수를 알수 없는 문자열 때 사용됨

nchar(n) : nchar(10) : 한글 10자

nvarchar2(n) : nchar(10) : 한글 10자

- 날짜 : date : BC 4712년 1월 1일 ~~~ 9999년 12월 31일까지저장, 년, 월, 일 시 분, 초 까지만 저장 가능

timestamp : insert (값이 들어오는 시스템의 시간) : 년, 월, 일, 시, 분,초 밀리센컨드 까지 저장

- LOB 데이터 타입 : 대량의 값을 저장, 바이너리 파일

CLOB : 문자를 대량으로 넣을수 있다. <== 글 내용

BLOB : mp3, jpeg, hwp 이진 데이터 파일

BFile : 대용량 파일 저장

\*/

--

drop table test10 ;

create table test10 (

id number(4) not null primary key , -- 중복된 넣을 수 없다. 정수 4자

n1 char(10) , -- 영문 10자, 한글 3자

n2 nchar(10) , -- 영문 10자 , 한글 10자

n3 varchar2(10) , -- 영문 10자, 한글 3자

n4 nvarchar2(10) -- 영문 10자 , 한글 10자

);

select \* from test10 ;

commit;

insert into test10 ( id , n1 , n2, n3, n4)

values ( 1111, 'abcdefghij', '가나다라마바사카차타' , 'abcdefghij', '가나다라마바사카차타' ) ;

insert into test10 ( id , n1 )

values ( 1112, '가나다') ;

/\*

제약 조건 : 테이블의 컬럼에 들어가는 키 , 데이터의 무결성 (결함없는데이터, 원하는 값만 )

Primary Key :

- 테이블의 컬럼에 1번만 넣을 수 있다.

- 두개 컬럼을 묶어서 PK를 넣을 수 있다.

- 테이블을 생성할때 반드시 PK 이 존재해야 한다. - Update, Delete 구문에서 PK 컬럼을 where 조건으로 사용함.

- 특정 컬럼에 중복 된 값을 넣지 못하도록 함.

- 반드시 not null 컬럼이어야함. null을 넣을 수 없다.

- index 가 자동으로 생성 된다. 컬럼의 검색을 빠르게 함.

- JOIN 시 ON 에서 많이 사용하는 키 컬럼.

Unique Key :

- 컬럼에 중복된 값을 넣지 못하도록 함.

- null을 넣을 수 있다. 단 1번만 넣을 수 있다. not null, null

- 하나의 테이블에 여러번 Unique Key 를 넣을 수 있다.

- index 가 자동으로 생성됨. JOIN 시 ON에 사용됨

Foreign Key :

- 다른 테이블(부모 테이블) 의 특정 컬럼을 참조 해서 값을 넣도록 함.

- Foreign Key 가 잠조하는 컬럼은 부모테이블의 Primary Key , Unique Key 를 참조함.

NOT NULL

- 컬럼에 NULL을 넣을 수 없도록 하는 제약 조건

CHECK

- 컬럼에 조건을 넣어서 내개 원하는 값만 넣을 수 있도록 함.

- 월 컬럼에 1 ~ 12 까지 넣을 수 있도록

default : 제약 조건은 아니지만 제약 조건처럼 사용됨

- 컬럼에 값을 넣지 않으면 default 로 설정된 값이 등록됨.

제약 조건을 출력하는 데이터 사전 : user\_constraints

select \* from user\_constraints where table\_name in ('테이블명') ;

\*/

-- 제약 조건 이름을 넣지 않고 테이블 생성 한 경우 : Oracle에서 제약조건이름을 랜덤하게 생성한다.

-- insert 시 오류가 발생할 경우 제약조건이름으로 오류난 컬럼을 찾기 힘들다.

create table member1 (

id varchar2(50) not null primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.

pass varchar2(50) not null,

addr varchar2(100) null ,

jumin char(13) null , -- 자릿수가 지정된 컬럼

phone varchar2(50) ,

age number(3) , -- 정수 3자리

weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리

) ;

desc member1 ;

insert into member1 (id , pass, addr, jumin, phone, age, weight)

values ( 'abc', '1234', '서울', '123456-789101', '010-1111-1111', 30 , 77.77);

commit ;

select \* from member1;

select \* from user\_constraints where table\_name in ('MEMBER1') ;

-- 테이블 생성시 제약 조건 이름 부여

create table member2 (

id varchar2(50) not null constraint PK\_MEMBER2\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.

pass varchar2(50) constraint NN\_MEMBER2\_PASS not null,

addr varchar2(100) null ,

jumin char(13) null , -- 자릿수가 지정된 컬럼

phone varchar2(50) ,

age number(3) , -- 정수 3자리

weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리

) ;

select \* from user\_constraints where table\_name in ('MEMBER2') ;

insert into member2 (id , addr, jumin, phone, age, weight)

values ( 'abcd', '서울', '123456-789101', '010-1111-1111', 30 , 77.77);

commit ;

/\* UNIQUE : 중복된 값을 넣을수 없다. null을 넣을 수 있다. 테이블에 여러번 넣을수 있다.

\*/

create table member3 (

id varchar2(50) not null constraint PK\_MEMBER3\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.

pass varchar2(50) constraint NN\_MEMBER3\_PASS not null,

addr varchar2(100) null ,

jumin char(13) null constraint U\_MEMBER3\_JUMIN unique , -- 중복되면 안됨

phone varchar2(50) not null constraint U\_MEMBER3\_PHONE unique , -- 중복되면 안됨

age number(3) , -- 정수 3자리

weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리

) ;

insert into member3 (id , pass, addr, jumin, phone, age, weight)

values ( 'abc', '1234', '서울', '123456-789102', '010-1111-1112', 30 , 77.77);

commit ;

-- CHECK 제약 조건 : 컬럼에 조건에 맞는 값만 넣을 수 있도록 함.

drop table member4 ;

create table member4 (

id varchar2(50) not null constraint PK\_MEMBER4\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.

pass varchar2(50) constraint NN\_MEMBER4\_PASS not null,

addr varchar2(100) null constraint CK\_MEMBER4\_ADDR check (addr in ('서울', '부산','대구')) ,

jumin char(13) null constraint U\_MEMBER4\_JUMIN unique , -- 중복되면 안됨

phone varchar2(50) not null constraint U\_MEMBER4\_PHONE unique , -- 중복되면 안됨

age number(3) constraint CK\_MEMBER4\_AGE check (age > 0 and age < 200) , -- 정수 3자리

gender char(1) constraint CK\_MEMBER4\_GENDER check ( gender in ('w', 'm')),

weight number (5,2) -- 실수 전체 5자리, 소숫점이하 2자리

) ;

insert into member4 (id , pass, addr, jumin, phone, age, weight, gender )

values ( 'abc', '1234', '대구', '123456-789102', '010-1111-1112', 100, 77.77, 'w');

commit ;

-- default : 제약조건이 아니다. 제약조건 이름을 부여 할 수 없다.

-- 값을 넣을때 값이 들어가고 값을 넣지 않을때 default 로 설정된 값이 들어간다.

create table member5 (

id varchar2(50) not null constraint PK\_MEMBER5\_ID primary key , -- 제약 조건 이름을 생략하면 Oracle에서 자동으로 지정함.

pass varchar2(50) constraint NN\_MEMBER5\_PASS not null,

addr varchar2(100) null constraint CK\_MEMBER5\_ADDR check (addr in ('서울', '부산','대구')) ,

jumin char(13) null constraint U\_MEMBER5\_JUMIN unique , -- 중복되면 안됨

phone varchar2(50) not null constraint U\_MEMBER5\_PHONE unique , -- 중복되면 안됨

age number(3) constraint CK\_MEMBER5\_AGE check (age > 0 and age < 200) , -- 정수 3자리

gender char(1) constraint CK\_MEMBER5\_GENDER check ( gender in ('w', 'm')),

weight number (5,2), -- 실수 전체 5자리, 소숫점이하 2자리

hiredate date default sysdate,

addr2 char(10) default '서울' ,

age2 number default 0

) ;

select \* from member5;

insert into member5 (id , pass, addr, jumin, phone, age, weight, gender )

values ( 'abc', '1234', '대구', '123456-789102', '010-1111-1112', 100, 77.77, 'w');

insert into member5 (id , pass, addr, jumin, phone, age, weight, gender , hiredate, addr2, age2 )

values ( 'abcd', '1234', '대구', '123456-789103', '010-1111-1113', 100, 77.77, 'w', '22/11/11', '광주', 30);

commit ;

-- 날짜 타입에 체크 제약 조건

CREATE TABLE emp55 (

e\_id NUMBER PRIMARY KEY, -- 아이디 컬럼

h\_date DATE CHECK (h\_date BETWEEN TO\_DATE('1981-01-01', 'YYYY-MM-DD') AND TO\_DATE('1981-12-31', 'YYYY-MM-DD'))

);

insert into emp55

values ( 10, '81/11/11');

commit ;

08.[Oracle] - [과제 제출용 테이블] - DML

**08. [Oracle] - [과제 제출용 테이블] - DML (Insert, Update, Delete)2025.04.15**

| -- 부서 테이블  CREATE TABLE DEPARTMENTS (  DEPT\_ID NUMBER PRIMARY KEY,  DEPT\_NAME VARCHAR2(50)  );  -- 사원 테이블  CREATE TABLE EMPLOYEES (  EMP\_ID NUMBER PRIMARY KEY,  EMP\_NAME VARCHAR2(50),  SALARY NUMBER,  DEPT\_ID NUMBER,  HIRE\_DATE DATE,  CONSTRAINT fk\_dept FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENTS(DEPT\_ID)  );  -- 프로젝트 테이블  CREATE TABLE PROJECTS (  PROJ\_ID NUMBER PRIMARY KEY,  PROJ\_NAME VARCHAR2(100),  EMP\_ID NUMBER,  START\_DATE DATE,  CONSTRAINT fk\_proj\_emp FOREIGN KEY (EMP\_ID) REFERENCES EMPLOYEES(EMP\_ID)  );  -- DEPARTMENTS  INSERT INTO DEPARTMENTS VALUES (10, '개발팀');  INSERT INTO DEPARTMENTS VALUES (20, '영업팀');  INSERT INTO DEPARTMENTS VALUES (30, '디자인팀');  -- EMPLOYEES  INSERT INTO EMPLOYEES VALUES (1001, '김철수', 3500000, 10, TO\_DATE('2020-01-10', 'YYYY-MM-DD'));  INSERT INTO EMPLOYEES VALUES (1002, '이영희', 4000000, 10, TO\_DATE('2019-03-12', 'YYYY-MM-DD'));  INSERT INTO EMPLOYEES VALUES (1003, '박지민', 2800000, 20, TO\_DATE('2021-06-01', 'YYYY-MM-DD'));  INSERT INTO EMPLOYEES VALUES (1004, '정국', 5000000, 30, TO\_DATE('2018-09-21', 'YYYY-MM-DD'));  INSERT INTO EMPLOYEES VALUES (1005, '최유리', 3200000, 20, TO\_DATE('2022-02-14', 'YYYY-MM-DD'));  -- PROJECTS  INSERT INTO PROJECTS VALUES (2001, '홈페이지 리뉴얼', 1001, TO\_DATE('2023-01-15', 'YYYY-MM-DD'));  INSERT INTO PROJECTS VALUES (2002, '고객관리 시스템', 1002, TO\_DATE('2023-03-01', 'YYYY-MM-DD'));  INSERT INTO PROJECTS VALUES (2003, '신제품 마케팅', 1003, TO\_DATE('2023-05-10', 'YYYY-MM-DD'));  INSERT INTO PROJECTS VALUES (2004, '브랜드 리디자인', 1004, TO\_DATE('2022-11-05', 'YYYY-MM-DD'));  INSERT INTO PROJECTS VALUES (2005, '데이터 분석 시스템', 1005, TO\_DATE('2023-07-01', 'YYYY-MM-DD'));  /\*  INSERT 관련  1. 새로운 부서 기획팀(DEPT\_ID=40)을 추가하세요.  2. 사원 김민수(EMP\_ID=1006)를 기획팀에 추가하세요. (급여 300만원, 입사일: 2024-01-01)  3. 프로젝트 사내 교육 시스템(PROJ\_ID=2006)을 김민수에게 배정하세요.  4. 새로운 사원 고유정(1007)을 영업팀에, 급여 310만원으로 추가하세요.  5. 정우성(1008)을 부서 없이 등록해보세요. (DEPT\_ID를 NULL로)  UPDATE 관련  6. 김철수의 급여를 380만원으로 수정하세요.  7. 박지민의 부서를 디자인팀으로 변경하세요.  8. 홈페이지 리뉴얼 프로젝트를 정국에게 재배정하세요.  9. 최유리의 이름을 최윤아로 변경하세요.  10. 모든 영업팀 직원의 급여를 10% 인상하세요.  DELETE 관련  11. 급여가 300만원 이하인 사원을 삭제하세요.  12. 기획팀의 부서를 삭제하려면 어떤 제약 조건을 고려해야 하나요? 삭제를 시도해보세요.  13. 사원 번호가 1008인 사람을 삭제하세요.  14. 김민수의 프로젝트를 삭제하세요.  15. 부서가 없는 사원을 모두 삭제하세요.  \*/  /\*  select \* from departments;  select \* from employees;  select \* from projects;  =========================문제 풀이=================================  INSERT 관련  1. 새로운 부서 기획팀(DEPT\_ID=40)을 추가하세요.  insert into departments (dept\_id, dept\_name)  values (40, '기획팀');  select \* from departments;  commit;  2. 사원 김민수(EMP\_ID=1006)를 기획팀에 추가하세요. (급여 300만원, 입사일: 2024-01-01)  insert into employees (emp\_id, emp\_name, salary, dept\_id, hire\_date)  values (1006, '김민수', 3000000, (select dept\_id from departments  where dept\_name = '기획팀'), to\_date('24/01/01', 'YYYY-MM-DD')  );  select \* from employees;  commit;  3. 프로젝트 사내 교육 시스템(PROJ\_ID=2006)을 김민수에게 배정하세요.  insert into projects (proj\_id, proj\_name, emp\_id, start\_date)  values (2006, '프로젝트 사내 교육 시스템', (select emp\_id from employees  where emp\_name = '김민수'), sysdate);  select \* from projects;  commit;  rollback;  4. 새로운 사원 고유정(1007)을 영업팀에, 급여 310만원으로 추가하세요.  insert into employees (emp\_id, emp\_name, salary, dept\_id, hire\_date)  values (1007, '고유정', 3100000, (select dept\_id from departments  where dept\_name = '영업팀'), sysdate);  select \* from employees;  commit;  5. 정우성(1008)을 부서 없이 등록해보세요. (DEPT\_ID를 NULL로)  insert into employees (emp\_id, emp\_name, salary, dept\_id, hire\_date)  values (1008, '정우성', null, null, sysdate);  select \* from employees;  commit;  UPDATE 관련  6. 김철수의 급여를 380만원으로 수정하세요.  update employees  set salary = 3800000  where emp\_name = '김철수';  select \* from employees;  commit;  7. 박지민의 부서를 디자인팀으로 변경하세요.  select \* from employees;  update employees  set dept\_id = (select dept\_id from departments  where dept\_name = '디자인팀')  where emp\_name = '박지민';    commit;  8. 홈페이지 리뉴얼 프로젝트를 정국에게 재배정하세요.  select \* from projects;  select \* from employees;  update projects  set emp\_id = (select emp\_id from employees  where emp\_name = '정국')  where proj\_name = '홈페이지 리뉴얼';    commit;  9. 최유리의 이름을 최윤아로 변경하세요.  select \* from employees;  update employees  set emp\_name = '최윤아'  where emp\_name = '최유리';    commit;  10. 모든 영업팀 직원의 급여를 10% 인상하세요.  select \* from departments;  select \* from employees;  select \* from projects;  update employees  set salary = salary \* 1.1  where dept\_id = (select dept\_id from departments  where dept\_name = '영업팀');  commit;  DELETE 관련  11. 급여가 300만원 이하인 사원을 삭제하세요.  select \* from departments;  select \* from employees;  select \* from projects;  -- projects 가 employee 를 참조 하므로 삭제 오류 발생  DELETE FROM employees;  where salary <=3000000 ;  -- 참조 되는 projects 테이블의 값을 먼저 삭제  DELETE FROM projects  WHERE emp\_id IN (  SELECT emp\_id FROM employees WHERE salary <= 3000000  );  delete from employees  where salary <= 3000000;  select \* from employees;  commit;  12. 기획팀의 부서를 삭제하려면 어떤 제약 조건을 고려해야 하나요? 삭제를 시도해보세요.  select \* from departments;  select \* from employees;  select \* from projects;  -- employees 의 dept\_id 가 departments 테이블의 dept\_id 을 참조 하므로 참조하는 레코드 먼저 삭제함  DELETE FROM employees  WHERE dept\_id = 40;  delete from departments  where dept\_name = '기획팀';  commit;  13. 사원 번호가 1008인 사람을 삭제하세요.  select \* from departments;  select \* from employees;  select \* from projects;  delete from employees  where emp\_id = 1008;  select \* from employees;  commit;  14. 김민수의 프로젝트를 삭제하세요.  select \* from departments;  select \* from employees;  select \* from projects;  delete from projects  where emp\_id = (select emp\_id from employees  where emp\_name = '김민수');  commit;  15. 부서가 없는 사원을 모두 삭제하세요.  delete from employees  where dept\_id is null;  commit;  참고:  -- 부모 테이블  CREATE TABLE departments (  dept\_id NUMBER PRIMARY KEY,  dept\_name VARCHAR2(50)  );  --자식테이블  CREATE TABLE employees (  emp\_id NUMBER PRIMARY KEY,  emp\_name VARCHAR2(50),  dept\_id NUMBER,  CONSTRAINT fk\_dept  FOREIGN KEY (dept\_id)  REFERENCES departments(dept\_id)  ON DELETE CASCADE  );  -- 사원 3명이 부서 10번에 소속되어 있다고 가정  DELETE FROM departments WHERE dept\_id = 10;  -- 결과:  -- → departments에서 10번 부서가 삭제됨  -- → employees에서 dept\_id가 10인 사원들도 자동 삭제됨  #### 이미 만들어진 테이블 에 적용하는 방법  1. 기존의 제약 조건 제거  ALTER TABLE employees DROP CONSTRAINT fk\_dept;  2. 기존 테이블의 제약 조건 추가  ALTER TABLE employees  ADD CONSTRAINT fk\_dept  FOREIGN KEY (dept\_id)  REFERENCES departments(dept\_id)  ON DELETE CASCADE;  \*/ |
| --- |

탭 10

docker run -d --name oracle-21c -p 10521:1521 -e ORACLE\_PASSWORD=1234 kite01/oracle-xe-21c