오라클 데이타베이스 ?
데이터베이스는 기본적으로 데이터를 저장하는 저장소의 역할을 한다.데이터베이스를 운영하려면 데이터베이스 운영 시스템이 필요하다. 데이터베이스 운영 시스템은 DBMS(DataBase Management System)라고 한다.

DBMS는 소프트웨어이다. DBMS는 하드웨어(컴퓨터 시스템)에 DB 생성작업을 진행한다. 이러한 DBMS를 기반으로 데이타베이스가 생성이 되고 관리된다.

DBMS에는 종류와 버전이 있고 ORACLE 12c가 ORACLE DBMS 12c버전을 가리킨다.ORACLE DBMS 12c는 RDBMS의 한 종류이다.

DBMS의 종류는 DBMS를 소프트웨어로 판매를 하는 회사별로 나누어 볼 수 있다.예를 들면 ORACLE DBMS, IBM DB2 DBMS, MICROSOFT SQL SERVER DBMS 등이 있다.현재 대부분의 DBMS들은 관계형 데이터베이스 관리시스템(RDBMS)으로 제공되고 있다.

관계형 데이터베이스 관리시스템=> RDBMS(RELATIONAL DATABASE MANAGEMENT SYSTEM) 관계형 데이터베이스를 관리하기 위해 사용되는 소프트웨어

(DBMS)=> RDBMS를 통해서 데이터베이스에서 데이터와 데이터 들간의 관계를 저장 관리한다.

ORDBMS (OBJECT RELATIONAL DATABASE MANAGEMENT SYSTEM)=> 데이터를 객체의 개념으로 관리하는 DBMS
ORACLE DBMS 12c는 RDBMS와 ORDBMS 기능 모두 지원한다.

관계형 데이터베이스 (RELATIONAL DATABASE)의 내용

1. 기본적으로 모든 데이터들이 테이블 구조를 통해 저장 관리. 여러개의 테이블들이 모여져서 관계형 데이터베이스의 내용을 구성하게 된다.

2. 테이블과 테이블 사이에 데이터들 간의 관계를 만들어주고 운용.
테이블 (TABLE) 구조 : 2차원 구조로 되어 있다.칼럼(COLUMN)과 레코드(RECORD, ROW)로 구성이 되어 있다.

데이터베이스 설계도 (관계형 모델링) : 테이블에 대한 설계작업
1. 논리적 설계도 (개념적 설계, 엔터티 모델링, 1차 설계도) -- 결과물 : ERD (ENTITY RELATIONSHIP DIAGRAM, 엔터티 관계 모델) - 1차 설계도– ERD를 그리는 방법은 여러가지가 있다. (BACHMAN방식, BAKER방식 등등)=> 순수하게 데이타와 데이타들간의 관계만을 보고 테이블 설계2. 물리적 설계도 (테이블 모델링, 2차 설계도) -- 결과물 : TABLE DIAGRAM - 2차 설계도=> 사용하고자 하는 데이터베이스의 종류에 따라서 논리적 설계도 편집
관계는 양방향성을 가진다관계는 프로세스(PROCESS)에서 생성된다.

부서데이터와 사원데이터 간의 관계는 부서와 사원간의 프로세스를 통해 여러 관계가 생성될 수 있다.예를 들면 사원이 부서에 배치된다는 프로세스를 통해 어떤 사원이 어떤 부서에 소속이 된다 라는 관계가 만들어진다.

관계형 데이터베이스는 테이블과 테이블 데이터들간의 관계를 관리해 주는 기능을 가지고 있는데 이때 관계에 대한 정보는 DETAIL TABLE 쪽에 FOREIGN KEY 컬럼을 이용해서 표시를 해 주게 된다. FOREIGN KEY 컬럼을 통해서 데이터들의 관계를 관리해 주는 데이타베이스가 관계형 데이터베이스라고 할 수 있다.

-- 테이블에는 KEY가 되는 데이터가 있다
KEY가 되는 되는 데이타란 대표값이 되는 값이다.
예를 들면, 대한 민국 국민의 데이타 중에 키가될 수 있는 값은 주민등록번호KEY가 되는 값이라는 것은 다른 데이타들과 비교해서 중복되지 않게 해당 데이터를구별할 때 사용할 수 있는 데이터가 PRIMARY KEY가 될 수 있는 데이터가 된다.

관계형 데이타베이스는 서로 관계가 있는 테이블들의 집합으로 되어 있는데서로 관계가 있는 테이블을 MASTER 또는 PARENT TABLE과 DETAIL 또는 CHILDTABLE 라는 개념을 가지고 일반적으로 MASTER 또는 PARENT TABLE에 있는 PRIMARY KEY에 해당하는 데이터가 DETAIL 또는 CHILD TABLE안에 데이터로 COPY되어서 만들어지는FOREIGN KEY 데이터가 두 개의 테이블 사이의 관계를 설명해 주는 역할을 하게 된다.

1: RECORD(ROW)
2: PRIMARY KEY COLUMN
3. 숫자 데이터를 가지는 일반 COLUMN
4. FOREIGN KEY COLUMN
5. FIELD, CELL
6. NULL 상태(데이타가 없는 상태)의 FIELD, CELL

SQL (STRUCTURED QUERY LANGUAGE)-- 문법을 가지고 있는 데이터베이스에 저장된 데이터를 조회,편집하는 언어

SQL의 종류DML(DATA MANIPULATION LANGUAGE)DDL(DATA DEFINITION LANGUAGE) : TABLE 생성 관리DCL(DATA CONTROL LANGUAGE) : DB 사용자 권한 관리 TRANSACTION CONTROL LANGUAGE : 트랜잭션 처리

SQL DEVELOPER, SQL PLUS TOOL에서 제공하는 명령어SHOW USER (SQL DEVELOPER 명령) 현재 어떤 DB USER로 로그인되어 있는지 확인하는 명령 SELECT * FROM TAB; 현재 사용자가 사용할 수 있는 테이블 목록 조회하는 SQL명령

DESC DEPARTMENTS (SQL DEVELOPER 명령) DEPARTMENTS 테이블안에 COLUMN구조정보 조회DESCRIBE DEPARTMENTS
SQL DEVELOPER 명령들은 축약어를 사용할 수 있다 .SQL 명령은 축약어 사용 불가능 (예 . SELECT => SEL (X))

SELECT *|{[DISTINCT] column [alias],...} – SELECT 절(CLAUSE)FROM table; – FROM 절(CLAUSE)

SELECT * FROM DEPARTMENTS;DEPARTMENTS 테이블에 저장되어 있는 모든 COLUMN과 RECORD들의 정보 조회

SELECT DEPARTMENT_ID, DEPARTMENT_NAME FROM DEPARTMENTS;
DEPARTMENTS 테이블에 저장되어 있는 DEPARTMENT_ID, DEPARTMENT NAME 컬럼의 데이터만 조회

SELECT DEPARTMENT_ID, DEPARTMENT_NAME FROM DEPARTMENTSWHERE DEPARTMENT_ID=10;
DEPARTMENTS 테이블에 저장되어 있는 DEPARTMENT_ID, DEPARTMENT NAME 컬럼의 데이터를 10번부서에 대해서만 조회

SQL DEVELOPER TOOL에서 2개이상의 SQL명령을 동시에 실행할 때는 각 명령문의 마지막 부분에 ;(세미콜론)을 반드시 설정해야 한다.하나의 SQL명령만 실행하는 경우에는 ;(세미콜론)을 문장끝에 있건 없건 상관없다.
SELECT * FROM EMPLOYEES;
SELECT department_id, location_idFROM departments;

SQL*PLUS TOOL에서는 모든 SQL명령의 끝에 반드시 ;(세미콜론)이 있어야 실행된다.

숫자 데이터에 대해서는 +, -. *, / 산술연산자 모두 사용가능 날짜 데이터에 대해서는 +, - 산술연산자 모두 사용가능
SELECT 3*5 FROM DUAL;
SELECT 3+5+NULL FROM DUAL;

열 alias(별칭)
SELECT last_name "   Name" , salary*12 "Annual Salary"FROM employees;

연결 연산자(CONCATENATION OPERATOR)
SELECT LAST_NAME,JOB_ID,last_name||job_id AS "Employees"FROM employees;
COL LAST_NAME FORMAT A12COL Employees FORMAT A20

SELECT LAST_NAME,JOB_ID,last_name||job_id AS "Employees"FROM employees;

-- 대체인용 연산자 (q)--
DUAL은 DUMMY TABLECOL TEST FORMAT A5
SELECT 'AA' "TEST" FROM DUAL; (O)
SELECT 'A'A' "TEST" FROM DUAL; (X)
SELECT 'A"A' "TEST" FROM DUAL; (O)
SELECT q'[A'A]' "TEST" FROM DUAL; (O)
SELECT q'!A'A!' "TEST" FROM DUAL; (O)
SELECT LAST_NAME,JOB_ID,last_name|| q'!A'A!'||job_id AS "Employees"FROM employees;
SELECT LAST_NAME,JOB_ID, last_name||' '|| q'!A'A!'||' '||job_id AS "Employees"FROM employees;

– department_id, job_id 조합된 결과에 대해 중복 데이터에서 하나만 DISPLAY해 주는 명령
SELECT department_id, job_id FROM employees;
SELECT distinct department_id, job_id FROM employees;

기본 날짜데이타의 형식은 데이터베이스별로 다르게 설정될 수 있고 이 형식은 DB관리자에 의해 관리됨.현재 사용중인 데이터베이스의 기본 날짜형식을 확인하기 위해서는 아래의 명령으로 확인 가능하다.
SELECT SYSDATE FROM DUAL; – 현재 사용중인 데이터베이스의 현재 날짜 정보 조회
날짜 데이터에 대해 조건을 줄때는 기본 날짜형식에 맞추어서 조건을 입력해야 한다.

날짜 형식은 사용자가 아래의 명령으로 변경할 수 있다.
ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';
SELECT SYSDATE FROM DUAL;
위의 명령으로 변경한 내용은 현재 사용자에게만 적용되고 현재 사용자가 LOGOUT하게 되면 다시 로그인하면 DB의 원래의 형식으로 바뀌게 된다.날짜를 변경하는 작업은 DB LOGIN할때마다 실행을 해 주어야 한다. SQL

DEVELOPER를 사용하는 경우에는 SQL DEVELOPER안에다가 사용자가 사용할 날짜형식을 고정적으로 셋업을 해 놓고 사용할 수 있다.

```
ALTER SESSION SET NLS_DATE_FORMAT='MM-DD-YYYY';
SELECT SYSDATE FROM DUAL;
SELECT last_name, HIRE_DATEFROM employeesWHERE hire_date =
'09-17-2003';
```

날짜데이타를 가지는 컬럼에 대해 조건을 줄때에는 현재 사용하고 있는 날짜형식에 맞추어서 조건값을 주어야 조회가능

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';-- 2004년 1월 1일
이후에 입사한 사원정보 조회
SELECT last_name, HIRE_DATEFROM employees WHERE HIRE_DATE >=
'2004-01-01';
-- 2004년 1월 1일 이전에 입사한 사원정보 조회
SELECT last_name, HIRE_DATE FROM employees WHERE HIRE_DATE <=
'2004-01-01';
-- LAST_NAME이 'W'이후의 문자로 시작하는 사원
SELECT last_name, HIRE_DATE FROM employees WHERE LAST_NAME >=
'W';
-- 2004년 1월 1일에 입사하지 않은 사원정보 조회
SELECT last_name, HIRE_DATE FROM employeesWHERE HIRE_DATE <>
'2004-01-01';
SELECT last_name, HIRE_DATE FROM employeesWHERE HIRE_DATE !=
'2004-01-01';
```

급여가 2500미만이거나 3500을 초과하는 사원 조회

```
SELECT last_name, salary FROM employeesWHERE salary NOT BETWEEN 2500
AND 3500 ;
-- 2002-01-01 과 2005-12-30 사이에 입사한 사원정보 조회
SELECT last_name, HIRE_DATE FROM employees WHERE HIRE_DATE
BETWEEN '2002-01-01' AND '2005-12-30' ;
-- LAST_NAME이 'H'에서 'L'사이의 문자로 시작하는 사원정보 조회
SELECT last_name FROM employees WHERE LAST_NAME BETWEEN 'H' AND
'L' ;
```

– MANAGER_ID 칼럼 (직속상관의 사번)이 100이나 101이나 201인 사원 조회
SELECT employee_id, last_name, salary, manager_id FROM employees WHERE manager_id IN (100, 101, 201) ;
SELECT employee_id, last_name, salary, manager_id FROM employeesWHERE manager_id=100 OR manager_id=101 OR manager_id=201 ;
– MANAGER_ID 칼럼 (직속상관의 사번)이 100이나 101이나 201이 아닌 사원 조회
SELECT employee_id, last_name, salary, manager_id FROM employees WHERE manager_id NOT IN (100, 101, 201) ;
SELECT employee_id, last_name, salary, manager_id FROM employees WHERE manager_id <>100 AND manager_id<>101 AND manager_id<>201 ;
SELECT employee_id, last_name, salary, manager_id FROM employees WHERE manager_id != 100 AND manager_id != 101 AND manager_id != 201 ;

– LIKE 연산자주로 문자나 날짜 데이터에 대해 사용하는 연산자이고 다음과 같은 경우에 사용한다.
– FIRST_NAME 컬럼에 S로 시작하는 이름을 가지고 있는 사원 조회
– FIRST_NAME 컬럼에 'S' 글자만 가지고 있는 사원도 조회
SELECT first_name FROM employees WHERE first_name LIKE 'S%' ;
– FIRST_NAME 컬럼에 S로 시작하면서 바로 뒤에 어떤 문자 하나가 나오는 이름(예:SA)을 가지고 있는 사원 조회–
FIRST_NAME 컬럼에 'S' 글자만 가지고 있는 사원도 조회 불가능
SELECT first_name FROM employees WHERE first_name LIKE 'S_';
– FIRST_NAME 컬럼에 'S_'라는 이름을 가지고 있는 사원 조회
SELECT first_name FROM employees WHERE first_name = 'S_';
– FIRST_NAME 컬럼에 'S%'라는 이름을 가지고 있는 사원 조회
SELECT first_name FROM employees WHERE first_name = 'S%';

SELECT first_name FROM employees WHERE first_name LIKE 'St_' ; -- St로 시작하고 다음에 어떤 하나의 문자가 나오는 사원조회ex) StA(O) , StAB (X)

SELECT first_name FROM employees WHERE first_name LIKE 'S____' ;=> FIRST_NAME이 S로 시작하고 뒤에 이어지는 문자의 갯수가 4개가 나오는 사원 검색
SELECT first_name FROM employees WHERE first_name LIKE 'S__%' ;=> FIRST_NAME이 S로 시작하고 반드시 뒤에 2개이상의 문자가 있는 사원 검색-- Sam (0) Sa(X) Samd (0) S (X)
SELECT first_nameFROM employeesWHERE first_name LIKE '_S_%' ;=> FIRST_NAME이 두번째 문자가 S로 되어 있고 반드시 'S'의 앞에는 반드시 1개의 문자가 있고 뒤에는 1개 이상의 문자를 가지고 있는 사원 검색

-- ESCAPE 식별자 : LIKE 연산자를 사용할 때 '_' 나 '%'를 문자로 검색하고자 할 때 사용.
SELECT first_name FROM employees WHERE first_name LIKE 'S_&_%' ESCAPE '&'; – '&' 를 ESCAPE식별자로 설정=> FIRST_NAME이 S로 시작하고 2번째문자는 어떤 문자든 하나의 문자가 나오고 3번째문자는 '_'가 들어가야 한다. 이후에는 문자가 나올 수도 안 나올수도 있다.-- Sa_ (0) Saa (X) Sb_y (o)
SELECT first_nameFROM employeesWHERE first_name LIKE 'S_#_#%'

ESCAPE '#';=> FIRST_NAME이 S로 시작하고 3번째문자는 '_'가 들어가야 하고 4번째문자는 '%'가 되어야 한다 Sa_%(O) Sa_A(X) Sa_%A (X)
SELECT first_nameFROM employeesWHERE first_name LIKE 'S_#_#%%' ESCAPE '#';Sa_%A (O)

SELECT first_nameFROM employeesWHERE first_name NOT LIKE '_S_%' ;=> FIRST_NAME이 두번째 문자가 S로 되어 있고 반드시 'S'의 앞에는 반드시 1개의 문자가 있고 뒤에는 1개 이상의 문자를 가지고 있지 않은 사원 검색

– NULL 연산자– MANAGER_ID 컬럼에 데이터가 NULL인 상태인 사원 조회– MANAGER가 없는 사원 조회
SELECT last_name, manager_id FROM employees WHERE manager_id IS NULL ; (O)

SELECT last_name, manager_id FROM employees WHERE manager_id = NULL ; (X)

– manager_id가 NULL이 아닌 사원 조회– MANAGER가 있는 사원 조회
SELECT last_name, manager_id FROM employees WHERE manager_id IS NOT NULL ; (O)
SELECT last_name, manager_id FROM employees WHERE manager_id <> NULL ; (X)
COL Result format a6SELECT 1+1||2 "Result" FROM DUAL;
SELECT 1+(1||2) "Result" FROM DUAL;
– ORDER BY 절 – 입사일자를 기준으로 해서 오름차순으로(오래된 입사일자에서 최근 입사일자 순으로 정렬)
SELECT last_name, job_id, department_id, hire_dateFROM employeesORDER BY hire_date;
– 입사일자를 기준으로 해서 내림차순으로(최근 입사일자에서 오래된 입사일자 순으로 정렬)
SELECT last_name, job_id, department_id, hire_dateFROM employeesORDER BY hire_date desc;
SELECT last_name, department_id, salaryFROM employeesORDER BY department_id , salary DESC;=> 1차적으로 department_id 데이타를 기준으로 해서 전체 사원 데이터를 오름차순 정렬2차적으로 같은 부서에 있는 사원들에 대해서 SALARY를 기준으로 해서 내림차순 정렬

– 사원 데이터를 사원번호를 기준으로 정렬한 결과에서 위에서 5개의 RECORD만 조회
SELECT employee_id, first_name FROM employees ORDER BY employee_id FETCH FIRST 5 ROWS ONLY;

– 사원 데이터를 급여를 기준으로 내림차순으로 정렬한 결과에서 상위 5개의 사원정보만 조회
SELECT employee_id, first_name, SALARY FROM employees ORDER BY SALARY DESCFETCH FIRST 5 ROWS ONLY;
– 사원 데이터를 사원번호를 기준으로 정렬한 결과에서 위에서 6번째 사원부터 5명의 사원정보를 조회

SELECT employee_id, first_nameFROM employeesORDER BY employee_idOFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;

치환변수DEFINE A – A 이름의 치환변수안에 어떤 값이 셋업이 되어있는지 확인UNDEFINE A – A 이름의 치환변수를 지운다.
SELECT employee_id, last_name, job_id, &&AFROM employeesORDER BY &A;

DEFINE A = 'DEPARTMENT_ID' – A 이름의 치환변수가 생성되고 안에 'DEPARTMENT_ID' 값을 미리 셋업을 해 두겠다– 아래의 &A에 대해서 별도로 셋업이 되지 않고 위에 정의된 내용이 재사용이 될 수도 있다.
SELECT employee_id, last_name, job_id, &AFROM employees ORDER BY &A;

함수(FUNCTION)는 DB안에 저장되어 실행되는 프로그램이다. 함수는 BUILT IN 함수 (오라클에서 기본적으로 제공하는 함수), 사용자가 개발해서 사용하는 CUSTOMIZED 함수(PL/SQL 프로그래밍 언어를 사용).
SQL명령문 안에서 사용되는 함수를 SQL 함수라고 한다. SQL함수에는 여러 종류들이 있다.
함수는 하나이상의 입력값(인수)을 받아서(입력값이 없는 함수도 있음) 항상 하나의 결과값을 만드는 프로그램.
SELECT SYSDATE FROM DUAL; – SYSDATE는 입력값을 가지지 않는 함수SELECT ROUND(12.345, 2) FROM DUAL; – 반올림 함수

– 테이블의 각 RECORD마다 실행되는 함수를 단일행 함수 (SINGLE ROW FUNCTION)이라고 한다.
SELECT EMPLOYEE_ID, SALARY, ROUND(SALARY,-2) "RESULT" FROM EMPLOYEES;
– 여러행 함수 (그룹함수, MULTIPLE ROW FUNCTION, GROUP FUNCTION)– 테이블안에 여러개의 RECORD의 데이터가 입력값이 되어 하나의 결과가 도출되는 함수
SELECT SUM(SALARY) FROM EMPLOYEES;

– 사원테이블에서 50번부서에 소속된 사원들의 급여의 총합계를 구한다.–
이렇게 여러개의 RECORD의 값을 이용해서 하나의 결과값을 만드는 함수를 여러행 함수 혹은 – 그룹함수라고 한다.
SELECT SUM(SALARY) "RESULT" FROM EMPLOYEESWHERE DEPARTMENT_ID = 50;

<< 단일행 함수 >> single row function

문자함수 : 문자 데이터에 대해 실행되는 함수

대소문자 변환 함수1. LOWER (문자를 소문자로 바꾸어 주는 함수)
SELECT LAST_NAME, LOWER(LAST_NAME) FROM EMPLOYEES WHERE DEPARTMENT_ID=60;
SELECT LAST_NAME, LOWER(LAST_NAME) FROM EMPLOYEESWHERE LAST_NAME = 'Hunold'; – 기본적으로 문자 데이터의 경우는 대소문자를 구분해서 조회됨
SELECT LAST_NAME, LOWER(LAST_NAME) FROM EMPLOYEESWHERE LAST_NAME = 'hunold'; (x)
SELECT LAST_NAME, LOWER(LAST_NAME) FROM EMPLOYEESWHERE LOWER(LAST_NAME) = 'hunold'; – 문자데이터의 비교시 LAST_NAME 컬럼의 데이타가 잠시 소문자로 변환되어서 비교되기 때문에 위의 결과는 정상적으로 조회가 가능해짐.

2.UPPER (문자를 대문자로 바꾸어 주는 함수)
SELECT LAST_NAME, UPPER(LAST_NAME) FROM EMPLOYEESWHERE DEPARTMENT_ID=60;
SELECT LAST_NAME, UPPER(LAST_NAME) FROM EMPLOYEESWHERE LAST_NAME = 'HUNOLD'; (X)
SELECT LAST_NAME, UPPER(LAST_NAME) FROM EMPLOYEESWHERE UPPER(LAST_NAME) = 'HUNOLD'; (O)

3.INITCAP : 첫번째 문자만 대문자로 하고 나머지 문자들은 소문자로 변환해주는 함수
SELECT INITCAP('ABCDEF') FROM DUAL;
SELECT INITCAP('ABC DEF') FROM DUAL;

문자 조작 함수
1.CONCATSELECT CONCAT('ABC','DEF') FROM DUAL;
SELECT FIRST_NAME, LAST_NAME, CONCAT(FIRST_NAME,LAST_NAME) NAMEFROM EMPLOYEES;SELECT FIRST_NAME, LAST_NAME, FIRST_NAME||LAST_NAME NAME FROMEMPLOYEES;-- CONCAT함수는 최대 2개의 값까지만 연결할 수 있다
SELECT CONCAT('ABC','DEF','PPP') FROM DUAL; (X)

\2. SUBSTR– 'ABCDEFG' 문자열에서 앞에서 3번째 위치에서 시작해서 2개의 문자를 빼서 보겠다.
SELECT SUBSTR('ABCDEFG',3,2) FROM DUAL;– 'ABCDEFG'문자열에서 뒤에서 3번째 위치에서 시작해서 2개의 문자를 빼서 보겠다.
SELECT SUBSTR('ABCDEFG',-3,2) FROM DUAL;– 'ABCDEFG'문자열에서 앞에서 3번째 위치에서 시작해서 뒤에 있는 모든 문자를 빼서 보겠다.
SELECT SUBSTR('ABCDEFG',3) FROM DUAL;– 'ABCDEFG'문자열에서 뒤에서 3번째 위치에서 시작해서 뒤에 있는 모든 문자를 빼서 보겠다.
SELECT SUBSTR('ABCDEFG',-3) FROM DUAL;– FIRST NAME안에 이름에서 두번째 문자가 'o'로 되어 있는 사원 조회
SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEESWHERE SUBSTR(FIRST_NAME,2,1) = 'o';

\3. LENGTH LAST_NAME 컬럼안에 저장되어 있는 실제 데이터의 크기를 조회
SELECT LAST_NAME, LENGTH(LAST_NAME) NAME FROM EMPLOYEESWHERE EMPLOYEE_ID=100;
SELECT LENGTH('A B ') FROM DUAL;SELECT LENGTH(NULL) FROM DUAL;

4.INSTRSELECT INSTR('ABCFEFF','F') FROM DUAL; -- 'F'가 처음 나오는 위치– 3번째 위치에서 검색을 시작해서 'OR' 문자가 2번째로 나오는 위치
SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) "Instring" from dual;

\5. LPAD(LEFT PADDING)

```
COL TEST FORMAT A15
SELECT EMPLOYEE_ID, SALARY, LPAD(SALARY,10,'*') TEST FROM
EMPLOYEESWHERE DEPARTMENT_ID=20;
```

```
COL TEST FORMAT A15
SELECT EMPLOYEE_ID,LAST_NAME,    LPAD(LAST_NAME,15,' ') "      TEST"
FROM EMPLOYEESWHERE DEPARTMENT_ID=20;
```

\6. RPAD (RIGHT PADDING)

```
SELECT EMPLOYEE_ID,SALARY,RPAD(SALARY,16,'*') TEST FROM
EMPLOYEESWHERE DEPARTMENT_ID=20;
SELECT EMPLOYEE_ID,SALARY,RPAD(SALARY,16,' ') TEST FROM
EMPLOYEESWHERE DEPARTMENT_ID=20;
```

\7. TRIMSELECT TRIM(LEADING 'A' FROM 'ABACFADC') FROM DUAL;

```
SELECT LAST_NAME,TRIM(LEADING 'A' FROM LAST_NAME) FROM
EMPLOYEESWHERE SUBSTR(LAST_NAME,1,1) = 'A';
SELECT TRIM(TRAILING 'A' FROM 'ABACFADCA') FROM DUAL;
SELECT TRIM(BOTH 'A' FROM 'ABACFADCA') FROM DUAL;
```

\8. REPLACESELECT REPLACE('ABACFADCA','A','Z') FROM DUAL;SELECT REPLACE('ABACFADCA','A','') FROM DUAL;SELECT REPLACE('ABACFADCA','A',NULL) FROM DUAL;SELECT REPLACE(SUBSTR('ABACFADCA',4,3),'A','') FROM DUAL;

```
SELECT CONCAT(CONCAT(last_name, q'[' s job category is ]'), job_id) "Job"
FROM employees WHERE SUBSTR(job_id, 4) = 'REP';
```

숫자함수

1. ROUND (반올림 함수)select ROUND(45.926, 2) from dual;select ROUND(45.926, 1) from dual;select ROUND(45.926, 0) from dual;select ROUND(45.926) from dual;select ROUND(45.926, -1) from dual;

\2. TRUNC (절삭 함수)select
TRUNC(45.926, 2) from dual;
select TRUNC(45.926, 1) from dual;
select TRUNC(45.926, 0) from dual;
select TRUNC(45.926) from dual;
select TRUNC(45.926, -1) from dual;
select CEIL(2.83) from dual; -- 2.83보다 크면서 가장 가까운 정수
select FLOOR(2.83) from dual; -- 2.83보다 작으면서 가장 가까운 정수
select MOD(1600, 300) from dual; -- 1600을 300으로 나눈 나머지 값을 보여준다.
-- RR FORMAT에 대한 해석현재 2022년 이라면
ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-RR';
SELECT * FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-30'; (2030년으로 해석)
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-70'; (1970년으로 해석)

-- 현재 연도가 1998년이라면ALTER SESSION SET
NLS_DATE_FORMAT='DD-MON-RR';
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-30'; (2030년으로 해석)

SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-70'; (1970년으로 해석)

-- 현재 연도가 1940년이라면ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-RR';
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-30'; (1930년으로 해석)
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-70'; (1870년으로 해석)

-- YY 형식을 사용하는 경우-- 현재 2022년 이라면
ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YY';
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-30'; (2030년으로 해석)
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-70'; (2070년으로 해석)-- 현재 연도가 1998년이라면
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-30'; (1930년으로 해석)
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-70'; (1970년으로 해석)

-- YYYY 형식을 사용하는 경우-- 현재 2021년 이라면
ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YYYY';
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-2030';
(2030년으로 해석)
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-2070';
(2070년으로 해석)
SELECT *FROM EMPLOYEESWHERE HIRE_DATE > '10-7월-1970';
(1970년으로 해석)

-- 현재 사용하고 있는 데이타베이스가 운영되고 있는 지역의 시간대와 현재시간
SELECT DBTIMEZONE, SYSDATE, SYSTIMESTAMP FROM DUAL;

– 현재 사용자의 컴퓨터가 있는 지역의 시간대와 현재시간
SELECT SESSIONTIMEZONE, CURRENT_DATE, CURRENT_TIMESTAMP
FROM DUAL;
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD
HH24:MI:SS';SELECT SYSDATE, CURRENT_DATE FROM DUAL;

– 날짜 데이터에 대해 + 혹은 - 연산을 실행 가능SELECT
CURRENT_TIMESTAMP+10 FROM DUAL; --현재날짜에서 10일 후의 날짜
SELECT CURRENT_TIMESTAMP-10 FROM DUAL; --현재날짜에서 10일 전의
날짜
SELECT CURRENT_TIMESTAMP+10/24 FROM DUAL; 10시간 후의 날짜와
시간
SELECT CURRENT_TIMESTAMP+15/60/24 FROM DUAL; 15분 후의 날짜와
시간
SELECT CURRENT_TIMESTAMP+10/60/60/24 FROM DUAL; 10초 후의 날짜와
시간

-- CURRENT_DATE - hire_date 는 현재날짜 기준으로 입사일자이후 몇일이
경과되었는지
SELECT last_name,CURRENT_DATE, HIRE_DATE, CURRENT_DATE -
hire_date "Days"FROM employees;

-- 현재날짜 기준으로 입사일자이후 몇주가 경과했는지
SELECT last_name,CURRENT_DATE,HIRE_DATE, (CURRENT_DATE -
hire_date)/7 "Weeks"FROM employees;
날짜함수--현재 날짜와 2021년 9월 30일 사이의 개월수
ALTER SESSION SET NLS_DATE_FORMAT='YYYY/MM/DD';
SELECT MONTHS_BETWEEN(CURRENT_DATE,'2021/09/30') FROM DUAL;

-- 현재 날짜 기준으로 10개월 후의 날짜
SELECT ADD_MONTHS(CURRENT_DATE,10) FROM DUAL;

-- 현재 날짜 기준으로 다음에 첫번째 금요일의 날짜
SELECT NEXT_DAY(CURRENT_DATE,'금요일') FROM DUAL;

-- 현재 날짜 기준으로 해당월의 마지막 날짜
SELECT LAST_DAY(CURRENT_DATE) FROM DUAL;
SELECT LAST_DAY('1977/02/01') FROM DUAL;
SELECT ROUND('2021/12/12','MONTH') FROM DUAL; (X)
SELECT ROUND(TO_DATE('2021/12/12','YYYY/MM/DD'),'MONTH') FROM DUAL;
SELECT ROUND(TO_DATE('2021/12/17','YYYY/MM/DD'),'MONTH') FROM DUAL;
SELECT ROUND(TO_DATE('2021/05/12','YYYY/MM/DD'),'YEAR') FROM DUAL;
SELECT ROUND(TO_DATE('2021/11/17','YYYY/MM/DD'),'YEAR') FROM DUAL;
SELECT TRUNC(TO_DATE('2021/12/12','YYYY/MM/DD'),'MONTH') FROM DUAL;
SELECT TRUNC(TO_DATE('2021/12/17','YYYY/MM/DD'),'MONTH') FROM DUAL;
SELECT TRUNC(TO_DATE('2021/12/17','YYYY/MM/DD'),'YEAR') FROM DUAL;

변환함수 (CONVERSION FUNCTION)

암시적 데이터 유형 변환 (자동변환) :
IMPLICIT DATA TYPE CONVERSION: 변환함수가 없어도 변환이 가능WHERE SALARY = 5000 (5000 값이 숫자로 인식이 되고 자동변환이 필요없게 된다)WHERE SALARY = '5000'– SALARY 칼럼을 숫자 데이터 유형을 데이터로 가지는 칼럼– 위의 조건이 정상적으로 실행되기 위해서는 컬럼의 데이터 유형과 비교되는 데이터의 데이터 유형이 일치해야 한다-- '5000'은 문자로 인식되기 때문에 내부적으로 숫자로 변환하는 작업이 자동 실행된다.– 문자 '5000' 이 숫자 5000으로 자동 변환이 실행된다

만약 기본 날짜형식이 'YYYY/MM/DD'로 되어 있을 경우WHERE HIRE_DATE = '2002/07/01' – '2002/07/01' 데이터는 문자 데이터로 인식– '2002/07/01' 문자 데이터가 기본 날짜형식과 일치한다면 자동으로 날짜 데이터로 변환 가능WHERE HIRE_DATE = '07/01/2002' – '07/01/2002' 데이터는 문자 데이터로

인식– '07/01/2002' 문자 데이터가 기본 날짜형식과 일치하지 않기 때문에 자동으로 날짜 데이터로 변환 불가능
이런 경우에는 아래와 같이 강제적으로 날짜형식에 맞추어서 날짜 데이터로 변환해야 한다. WHERE HIRE_DATE = TO_DATE('07/01/2002','MM/DD/YYYY")

– 수동 변환(명시적 변환) 실행되어서 조건 검색이 가능해짐.
아래의 경우는 자동변환이 되는 경우
WHERE LAST_NAME = 12345;=> WHERE LAST_NAME = '12345'; – 자동변환 가능SELECT 1 + '1' FROM dual; '1' -> 1 (문자->숫자 자동 변환)SELECT 1 || '1' FROM dual; 1 ->'1' (숫자->문자 자동 변환)

명시적 데이터 유형 변환 (수동 변환) : EXPLICIT DATA CONVERSION변환함수의 예제
1. TO_CHAR : 숫자 데이터를 문자로 바꿀때숫자를 원하는 포맷에 맞게 Display 할 때 사용 (숫자 -> 포맷있는 문자열)
SELECT last_name,SALARY, TO_CHAR(salary, '$99,999,999.99')FROM employees;
SELECT last_name,SALARY, TO_CHAR(salary, '$00,000,000.00')FROM employees;

-- DB 시스템에 셋업되어 있는 기본 CURRECY SIGN(LOCAL CURRENCY SIGN)으로 나온다.– 기본 CURRENCY SIGN(LOCAL CURRENCY SIGN)은 사용자가 설정해서 사용가능
SELECT TO_CHAR(12345678.9,'L000,000,000.00') FROM DUAL;

날짜 데이터를 문자로 바꿀때 (날짜 데이터의 DISPLAY되는 FORMAT을 바꾸고 싶을때)날짜 정보를 원하는 포맷에 맞게 Display시 사용 (날짜포맷 있는 문자열)
SELECT TO_CHAR(CURRENT_DATE,'YEAR/MONTH/DDspth "OF" HH:MI:SS AM') FROM DUAL;
SELECT HIRE_DATE,TO_CHAR(HIRE_DATE,'YYYY/MM/DDSPTH HH24:MI:SS') HIREDATE FROM EMPLOYEES;2. TO_NUMBER : 문자 데이터를 숫자 데이터로 변환해주는 단일행 함수 포맷있는 문자열을 숫자로 변환 시 사용

SELECT 1 FROM dualWHERE '$12,345.67' = 12345.67 (X)

SELECT 1 FROM dualWHERE TO_NUMBER('$12,345.67', '$99,999.99') = 12345.67 (O)
SELECT 1 FROM dualWHERE TO_NUMBER('$12,345.67', '$00,000.00') = 12345.67 (O)
SELECT '$684,453.90' + 1000 FROM DUAL; (X)
SELECT TO_NUMBER('$684,453.90','$999,999.99') FROM DUAL; (O)
SELECT TO_NUMBER('$684,453.90','$999,999.99') + 1000 FROM DUAL; (O)

\3. TO_DATE : 문자 데이터를 날짜 데이터로 변환시키는 함수
SELECT SYSDATE - '03--15///2020' FROM DUAL; (X)
SELECT SYSDATE - TO_DATE('03--15///2020','MM--DD///YYYY') FROM DUAL; (O)
ALTER SESSION SET NLS_DATE_FORMAT='YYYY/MM/DD';
SELECT EMPLOYEE_ID,HIRE_DATE FROM EMPLOYEESWHERE HIRE_DATE > '2005/05/22'; --- 자동변환 (O)
SELECT EMPLOYEE_ID,HIRE_DATE FROM EMPLOYEESWHERE HIRE_DATE > '05--22///2005'; (X)
SELECT EMPLOYEE_ID,HIRE_DATE FROM EMPLOYEESWHERE HIRE_DATE > TO_DATE('05--22///2005','MM--DD///YYYY'); (O)

– fx 의 사용법
SELECT SYSDATE - TO_DATE('03-- 15///2020','MM--DD///YYYY') FROM DUAL; (O)
SELECT SYSDATE - TO_DATE('03-- 15///2020','fxMM--DD///YYYY') FROM DUAL; (X)
SELECT SYSDATE - TO_DATE('03-- 15///2020','fxMM-- DD///YYYY') FROM DUAL; (O)

일반함수
1.NVL : NULL이 있는 경우에 NULL이 아닌 다른 값으로 변환시켜주는 함수

SELECT EMPLOYEE_ID, COMMISSION_PCT, NVL(COMMISSION_PCT,0) COMMFROM EMPLOYEESWHERE DEPARTMENT_ID=60;
SELECT EMPLOYEE_ID, SALARY, COMMISSION_PCT, SALARY*COMMISSION_PCT COMMFROM EMPLOYEESWHERE DEPARTMENT_ID=60;
SELECT EMPLOYEE_ID,SALARY, COMMISSION_PCT, SALARY* NVL(COMMISSION_PCT,1) COMMFROM EMPLOYEESWHERE DEPARTMENT_ID=60;
SELECT EMPLOYEE_ID, COMMISSION_PCT,NVL(COMMISSION_PCT, 'no commission') COMMFROM EMPLOYEESWHERE DEPARTMENT_ID=60; (x)
SELECT EMPLOYEE_ID, COMMISSION_PCT, NVL(TO_CHAR(COMMISSION_PCT), 'no commission') COMM FROM EMPLOYEESWHERE DEPARTMENT_ID=60;(O)

\2. NVL2-- commission_pct 값이 null이 아니면 'SAL+COMM' 라고 PRINT하고-- commission_pct 값이 null이면 'SAL'이라고 PRINT해라.
SELECT last_name, salary, commission_pct, NVL2(commission_pct, 'SAL+COMM', 'SAL') incomeFROM EMPLOYEES;

\3. NULLIF-- FIRST_NAME 데이터의 크기와 LAST_NAME의 데이터의 크기를 비교해서-- 만약에 같은 값인 경우에는 NULL로 표시하고-- 만약에 다른 값인 경우에는 앞에 있는 데이터 (FIRST_NAME의 데이터의 크기)-- 크기를 PRINT해라
SELECT first_name, LENGTH(first_name) "expr1", last_name, LENGTH(last_name)"expr2", NULLIF(LENGTH(first_name), LENGTH(last_name)) result FROM employees;
SELECT NULLIF(1,2) RESULT FROM DUAL;SELECT NULLIF(1,1) RESULT FROM DUAL;

\4. COALESCE(인수1,인수2,인수3,인수4, …….)-- 인수1의 결과가 NULL이 아니면 인수1의 결과를 PRINT하고 실행을 종료.만약에 인수1의 결과가 NULL이면 인수2를 확인해서 인수2의 결과가 NULL이 아니면인수2의 결과를 PRINT하고 실행을 종료. 만약 인수2의 결과가 NULL이면 인수3의 결과를 확인하고인수3의 결과가 NULL이 아니면 인수3의 결과를 PRINT하고 실행을 종료. 인수3의 결과가NULL이면 인수4를 확인해서 인수4 결과가 NULL이 아니면 인수4의 결과를PRINT하고 ……..-- (salary+(commission_pct*salary) 의 결과가 NULL이 아니면 해당 결과를 PRINT하고-- (salary+(commission_pct*salary) 의 결과가 NULL이면 salary+2000의 결과를PRINT해라 salary+2000의 결과가 NULL인 경우에는 NULL이 나온다.

SELECT last_name, salary, commission_pct,
COALESCE((salary+(commission_pct*salary)), salary+2000) "New Salary"
FROMEmployees;

-- (salary+(commission_pct*salary) 의 결과가 NULL이 아니면 해당 결과를 PRINT하고-- (salary+(commission_pct*salary) 의 결과가 NULL이면 salary+2000의 결과를PRINT해라 salary+2000의 결과가 NULL인 경우에는 'no income' 이 print된다.

SELECT last_name, salary, commission_pct,
COALESCE((salary+(commission_pct*salary)), salary+2000,'no income') "New Salary"FROM employees; (x)

SELECT last_name, salary, commission_pct,
COALESCE((TO_CHAR(salary+(commission_pct*salary))),
TO_CHAR(salary+2000),'no income') "New Salary"FROM employees; (O)

– COMMISSION_PCT 컬럼에 NULL이 있는 사원은 제외하고 NULL이 아닌 값을 가지고 있는 사원들에대한 평균을 구하게 된다.
SELECT AVG(commission_pct) -- .2+.3+.2+.15/4FROM employees;

– EMPLOYEES 테이블에 있는 모든 사원이 평균값을 계산하는데 사용된다.–
NULL이 있는 사원도 포함해서 계산하게 된다
SELECT AVG(NVL(commission_pct, 0)) -- 0+0+....+.2+.3+.2+.15/20FROM
employees;

SELECT department_id, job_id, SUM(salary) –4
FROM employees –1
WHERE department_id > 40 –2
GROUP BY department_id, job_id –3
ORDER BY department_id; –5

SELECT department_id, COUNT(last_name)FROM employeesGROUP BY
department_id;
SELECT department_id, job_id, COUNT(last_name)FROM employeesGROUP BY
department_id, job_id;
SELECT department_id, job_id, COUNT(*)FROM employeesGROUP BY
department_id, job_id;

SELECT department_id, AVG(salary)FROM employees –1
WHERE AVG(salary) > 8000 (x)
GROUP BY department_id;

SELECT department_id, MAX(salary) –4
FROM employees –1
GROUP BY department_id –2
HAVING MAX(salary)>10000 ; –3

SELECT job_id, SUM(salary) PAYROLL –5
FROM employees –1
WHERE job_id NOT LIKE '%REP%' –2
GROUP BY job_id –3
HAVING SUM(salary) > 13000 –4
ORDER BY SUM(salary); –6

–일단 부서별로 데이터 그룹을 만들어서 평균급여 계산후 평균급여중에 가장 큰 평균급여를 조회
SELECT MAX(AVG(salary))FROM employeesGROUP BY department_id;

– GROUP BY 절에서 열 alias를 사용할 수 없습니다
SELECT department_id DEPTID, MAX(salary)
FROM employees
GROUP BY DEPTID (x)
HAVING MAX(salary)>10000 ;

SELECT department_id DEPTID, JOB_ID, MAX(salary)
FROM employees
GROUP BY department_id,JOB_ID
HAVING MAX(salary)>10000
ORDER BY DEPARTMENT_ID, JOB_ID DESC ;

조인 (JOIN): 두 개 이상의 테이블에서 서로 연관성이 있는 RECORD들을 하나의 RECORD로묶어서 조회를 하는 방법 (하나의 SELECT문으로 여러 테이블의 데이터들을 조회): 이 때 두 개 이상의 테이블에서 연관성을 가지는 컬럼을 이용해서 조인을 실행: 예를 들면 부서테이블에서 부서번호 컬럼과 사원 테이블의 부서번호 컬럼과 비교해서같은 부서 번호를 가지고 있는 부서정보과 사원정보를 묶어서 하나의 레코드로 조회할 수 있게 할 수 있다.: 조인할 수 있는 테이블의 조건은 테이블간에 관계(RELATION)가 있다면 기본적으로조인 가능한 테이블로 볼 수 있다. 하지만 예외적으로 서로 관계가

없는 테이블 사이에서도 비교할 수 있는 컬럼이 존재한다면 조인이 가능할 수 있다.

조인방법(조인 문법) : 조인 문법은 오라클 STYLE의 문법이 있고 ANSI 표준 문법(1999년부터 계속적으로 업데이트되고 있는 표준 문법)이 있다.

조인의 종류
1. EQU JOIN
2. NON EQU JOIN
3. SELF JOIN
4. OUTER JOIN

\1. EQU JOIN 두 개이상의 테이블의 연관성을 가지고 있는 컬럼의 데이터들을 "=" 비교연산자를 이용해서 조인을 하는 방식
– 부서에 소속되어 있는 사원의 정보와 해당 부서의 정보 같이 조회
SELECT DEPARTMENTS.DEPARTMENT_ID, DEPARTMENTS.DEPARTMENT_NAME, EMPLOYEES.EMPLOYEE_ID, EMPLOYEES.LAST_NAME, EMPLOYEES.DEPARTMENT_ID FROM DEPARTMENTS, EMPLOYEES WHERE DEPARTMENTS.DEPARTMENT_ID = EMPLOYEES.DEPARTMENT_ID; (조인식)

SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID, E.LAST_NAME, E.DEPARTMENT_ID FROM DEPARTMENTS D, EMPLOYEES EWHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID;

– 20번부서와 30번부서에 소속되어 있는 사원의 정보와 해당 부서의 정보 같이 조회
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID, E.LAST_NAME, E.DEPARTMENT_ID FROM DEPARTMENTS D, EMPLOYEES EWHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID AND D.DEPARTMENT_ID IN (20,30);

-- 부서정보와 부서의 관리자 정보 조회
SELECT E.EMPLOYEE_ID, E.LAST_NAME, D.DEPARTMENT_ID, D.DEPARTMENT_NAME, D.MANAGER_IDFROM DEPARTMENTS D, EMPLOYEES EWHERE D.MANAGER_ID = E.EMPLOYEE_ID;

– 사원중에 자신이 속해 있는 부서의 관리자와 자신의 직속상관이 같은 사원
SELECT E.EMPLOYEE_ID, E.LAST_NAME, E.MANAGER_ID, D.DEPARTMENT_ID, D.MANAGER_IDFROM DEPARTMENTS D, EMPLOYEES EWHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID AND D.MANAGER_ID = E.MANAGER_ID;

– 어떤 도시에 어떤 부서가 위치하고 있는지 조회
SELECT L.CITY, L.LOCATION_ID, D.LOCATION_ID, D.DEPARTMENT_NAMEFROM LOCATIONS L, DEPARTMENTS DWHERE L.LOCATION_ID = D.LOCATION_ID;
SELECT E.LAST_NAME, D.DEPARTMENT_NAME, L.CITY, C.COUNTRY_NAMEFROM EMPLOYEES E, DEPARTMENTS D, LOCATIONS L, COUNTRIES CWHERE E.DEPARTMENT_ID = D.DEPARTMENT_ID AND D.LOCATION_ID = L.LOCATION_IDAND L.COUNTRY_ID = C.COUNTRY_ID ;

\2. NON EQU JOIN컬럼을 비교시 = 를 사용하지 않고 다른 연산자(예를 들면 BETWEEN AND ..)를 이용해서 조인
JOBS 테이블: 직책(JOB_ID)별로 받을 수 있는 급여의 상한선과 하한선을 기록한 테이블; 어떤 사원이 현재 받고 있는 급여가 어떤 직책에서 받을 수 있는 상한선과 하한선사이의 급여를 받고 있는지 조회
SELECT E.EMPLOYEE_ID, E.LAST_NAME, E.SALARY, J.JOB_ID, J.JOB_TITLE, J.MIN_SALARY, J.MAX_SALARYFROM EMPLOYEES E, JOBS JWHERE E.SALARY BETWEEN J.MIN_SALARY AND J.MAX_SALARYAND E.EMPLOYEE_ID=201;

\3. SELF JOIN -- 하나의 테이블 안에 서로 관련성을 가지고 있는 RECORD들이 존재하는 경우– 100번사원을 직속상관으로 두고 있는 부하직원 정보 조회

SELECT M.EMPLOYEE_ID, M.LAST_NAME, E.EMPLOYEE_ID, E.LAST_NAME, E.MANAGER_IDFROM EMPLOYEES M, EMPLOYEES EWHERE M.EMPLOYEE_ID = E.MANAGER_ID AND M.EMPLOYEE_ID=100;
4.OUTER JOIN-- EQU JOIN을 실행하면서 EQU JOIN시 조회가 안되는 데이터도 같이 조회
– 부서배치가 안되어 있는 사원조회
SELECT EMPLOYEE_ID,LAST_NAME,DEPARTMENT_IDFROM EMPLOYEESWHERE DEPARTMENT_ID IS NULL;

-- 부서배치가 안되어 있는 사원 포함 조회-- 일단 사원테이블의 데이타 모두 조회하고 그 다음에 관련되는 부서 데이터 검색-- 기본적으로 EQU JOIN을 실행하고 여기에 부서배치가 안되어 있는 사원을 -- 추가적으로 조회 가능– 사원이 배치되어 있지 않은 부서도 조회되지 않는다
SELECT DEPARTMENTS.DEPARTMENT_ID, DEPARTMENTS.DEPARTMENT_NAME, EMPLOYEES.EMPLOYEE_ID, EMPLOYEES.LAST_NAME, EMPLOYEES.DEPARTMENT_ID FROM DEPARTMENTS, EMPLOYEES WHERE DEPARTMENTS.DEPARTMENT_ID(+) = EMPLOYEES.DEPARTMENT_ID;

-- 부서 테이블에 사원이 배치되어 있지 않은 부서도 조회-- 일단 부서테이블의 데이타 모두 조회하고 그 다음에 관련되는 사원 데이터 검색– 추가적으로 사원이 배치되어 있지 않은 부서도 같이 조회– 부서에 배치되어 있지 않은 사원(GRANT)은 조회되지 않음
SELECT D.DEPARTMENT_ID,D.DEPARTMENT_NAME,E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D,EMPLOYEES EWHERE D.DEPARTMENT_ID=E.DEPARTMENT_ID(+);

-- 부서 테이블에 사원이 배치되어 있지 않은 부서와 부서배치가 안되어 있는 사원 모두 조회하려고 할 경우– 오라클 스타일의 문법으로는 불가능– ANSI 표준문법으로는 실행 가능

```
SELECT
D.DEPARTMENT_ID,D.DEPARTMENT_NAME,E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D,EMPLOYEES EWHERE
D.DEPARTMENT_ID(+)=E.DEPARTMENT_ID(+); (X)
```

-- ANSI 표준 문법

1. NATURAL JOIN (EQU JOIN의 한 방법)-- 두개의 테이블에서 이름이 같고 데이터유형이 같은 컬럼을 찾아서 자동으로 EQU 조인하는 방식

```
SELECT DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,
E.LAST_NAMEFROM DEPARTMENTS D NATURAL JOIN EMPLOYEES E;
```

– 위의 조회결과와 아래의 결과는 같다.

```
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,
E.LAST_NAMEFROM DEPARTMENTS D, EMPLOYEES EWHERE
D.DEPARTMENT_ID = E.DEPARTMENT_ID AND
D.MANAGER_ID=E.MANAGER_ID;
```

– 3개이상의 테이블도 연결 가능

```
SELECT L.CITY, DEPARTMENT_ID, D.DEPARTMENT_NAME,
E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D NATURAL JOIN
EMPLOYEES E NATURAL JOIN LOCATIONS L;
```

– 위의 조회결과와 아래의 결과는 같다.

```
SELECT L.CITY, D.DEPARTMENT_ID, D.DEPARTMENT_NAME,
E.EMPLOYEE_ID, E.LAST_NAMEFROM DEPARTMENTS D, EMPLOYEES E,
LOCATIONS LWHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID AND
D.MANAGER_ID=E.MANAGER_IDAND D.LOCATION_ID = L.LOCATION_ID;
```

2. USING을 이용하는 방법 (EQU JOIN의 한 방법)-- 조인되는 두 개의 테이블에 이름이 같은 컬럼끼리만 조인가능

```
SELECT DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,
E.LAST_NAME,E.MANAGER_IDFROM DEPARTMENTS D JOIN EMPLOYEES
E USING (DEPARTMENT_ID);
```

SELECT DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID, E.LAST_NAME,E.MANAGER_IDFROM DEPARTMENTS D INNER JOIN EMPLOYEES E USING (DEPARTMENT_ID);
– 위의 조회결과와 아래의 결과는 같다.
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID, E.LAST_NAME,E.MANAGER_IDFROM DEPARTMENTS D , EMPLOYEES E WHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID;

SELECT DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID, E.LAST_NAME,MANAGER_IDFROM DEPARTMENTS D JOIN EMPLOYEES E USING (DEPARTMENT_ID,MANAGER_ID);
– 위의 조회결과와 아래의 결과는 같다.
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID, E.LAST_NAME,E.MANAGER_IDFROM DEPARTMENTS D , EMPLOYEES E WHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID AND D.MANAGER_ID=E.MANAGER_ID;

– 3개이상의 테이블도 연결 가능
SELECT L.CITY, DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D
JOIN EMPLOYEES E USING (DEPARTMENT_ID,MANAGER_ID)
JOIN LOCATIONS L USING (LOCATION_ID);

SELECT l.city, d.department_nameFROM locations l JOIN departments dUSING (d.location_id) (x)WHERE d.location_id = 1400 and l.city = 'Toronto';(x) – using절에서 사용되는 컬럼에는 alias를 주면 안된다.

SELECT l.city, d.department_nameFROM locations l JOIN departments dUSING (D.location_id) (X)WHERE location_id = 1400;

SELECT l.city, d.department_name, d.location_id (x)FROM locations l JOIN departments dUSING (location_id) WHERE location_id = 1400;
아래와 같이 실행하면 실행된다.SELECT l.city, d.department_nameFROM locations l JOIN departments dUSING (location_id)WHERE location_id = 1400 and l.city = 'Toronto';

\3. ON을 이용하는 방법 (EQU JOIN, NON EQU JOIN, SELF JOIN 모두 가능)
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D JOIN EMPLOYEES EON (D.DEPARTMENT_ID=E.DEPARTMENT_ID AND D.MANAGER_ID=E.MANAGER_ID);

SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D INNER JOIN EMPLOYEES EON (D.DEPARTMENT_ID=E.DEPARTMENT_ID AND D.MANAGER_ID=E.MANAGER_ID);

SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D JOIN EMPLOYEES EON D.DEPARTMENT_ID=E.DEPARTMENT_ID AND D.MANAGER_ID=E.MANAGER_ID;

– 위의 조회결과와 아래의 결과는 같다.
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.EMPLOYEE_ID,E.LAST_NAMEFROM DEPARTMENTS D, EMPLOYEES EWHERE D.DEPARTMENT_ID=E.DEPARTMENT_ID AND D.MANAGER_ID=E.MANAGER_ID;

SELECT E.LAST_NAME, D.DEPARTMENT_NAME, L.CITY, C.COUNTRY_NAMEFROM DEPARTMENTS D JOIN EMPLOYEES EON (D.DEPARTMENT_ID = E.DEPARTMENT_ID)JOIN LOCATIONS L ON

(D.LOCATION_ID = L.LOCATION_ID)JOIN COUNTRIES C ON (L.COUNTRY_ID = C.COUNTRY_ID)**AND** D.DEPARTMENT_ID < 40;
SELECT E.LAST_NAME, D.DEPARTMENT_NAME, L.CITY, C.COUNTRY_NAMEFROM DEPARTMENTS D JOIN EMPLOYEES EON (D.DEPARTMENT_ID = E.DEPARTMENT_ID)JOIN LOCATIONS L ON (D.LOCATION_ID = L.LOCATION_ID)JOIN COUNTRIES C ON (L.COUNTRY_ID = C.COUNTRY_ID)**WHERE** D.DEPARTMENT_ID < 40;
– 위의 조회결과와 아래의 결과는 같다.
SELECT C.COUNTRY_ID, C.COUNTRY_NAME, L.LOCATION_ID, L.CITY, D.DEPARTMENT_ID, D.LOCATION_ID, D.DEPARTMENT_NAME, E.DEPARTMENT_ID, E.LAST_NAMEFROM EMPLOYEES E, DEPARTMENTS D, LOCATIONS L, COUNTRIES CWHERE E.DEPARTMENT_ID = D.DEPARTMENT_IDAND D.LOCATION_ID = L.LOCATION_IDAND L.COUNTRY_ID = C.COUNTRY_ID AND D.DEPARTMENT_ID < 40;

\3. OUTER JOIN

– LEFT OUTER JOIN – 부서배치가 안되어 있는 사원정보 포함 조회 SELECT e.last_name, e.department_id, d.department_nameFROM employees e LEFT OUTER JOIN departments dON (e.department_id = d.department_id) ;

– 위의 조회결과와 아래의 결과는 같다.SELECT e.last_name, e.department_id, d.department_nameFROM employees e , departments dWHERE e.department_id = d.department_id(+) ;

– RIGHT OUTER JOIN – 사원배치가 안되어 있는 부서정보 포함 조회 SELECT e.last_name, d.department_id, d.department_nameFROM employees e RIGHT OUTER JOIN departments dON (e.department_id = d.department_id) ;

– 위의 조회결과와 아래의 결과는 같다.SELECT e.last_name, e.department_id, d.department_nameFROM employees e , departments dWHERE e.department_id(+) = d.department_id;

– FULL OUTER JOIN– 사원배치가 안되어 있는 부서정보와 부서배치가 안되어 있는 사원정보 포함 조회가능SELECT e.last_name, d.department_id, d.department_nameFROM employees e FULL OUTER JOIN departments dON (e.department_id = d.department_id) ;

– 아래의 문법은 실행 불가능 .SELECT e.last_name, e.department_id, d.department_nameFROM employees e , departments dWHERE e.department_id(+) = d.department_id(+); (X)

– INNER JOIN OUTER JOIN 방식이 아닌 다른 모든 조인 방식은 INNER JOINEQU JOIN, NON EQU JOIN, SELF JOIN들(USING이나 ON을 이용하는 JOIN)이 모두 INNER JOIN 방식

<<문제 1>> JOIN과 그룹함수가 같이 사용

DEPARTMENT_ID DEPARTMENT_NAME AVG_SAL---------- ------------------------------ ---------- 10 Administration 4400 20 Marketing 9500 50 Shipping 3500

AVG_SAL : 부서별 평균임금 (단, 10의 자리에서 반올림)정렬순서 : 부서코드 기준 오름차순

답안>>

SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, ROUND(AVG(E.SALARY),-1) AVG_SALFROM DEPARTMENTS D, EMPLOYEES EWHERE D.DEPARTMENT_ID=E.DEPARTMENT_ID AND D.DEPARTMENT_ID IN (10,20,50)GROUP BY D.DEPARTMENT_ID, D.DEPARTMENT_NAMEORDER BY D.DEPARTMENT_ID;

SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, ROUND(AVG(E.SALARY),-1) AVG_SALFROM DEPARTMENTS D JOIN EMPLOYEES EON D.DEPARTMENT_ID=E.DEPARTMENT_ID AND D.DEPARTMENT_ID IN (10,20,50)GROUP BY D.DEPARTMENT_ID, D.DEPARTMENT_NAMEORDER BY D.DEPARTMENT_ID;

SELECT DEPARTMENT_ID, D.DEPARTMENT_NAME, ROUND(AVG(E.SALARY),-1) AVG_SALFROM DEPARTMENTS D JOIN EMPLOYEES EUSING (DEPARTMENT_ID) WHERE DEPARTMENT_ID IN (10,20,50)GROUP BY DEPARTMENT_ID, D.DEPARTMENT_NAMEORDER BY DEPARTMENT_ID;

<< 문제 2 >> 그룹함수와 CASE문을 이용하는 문제

다음과 같은 결과가 출력되도록 Employees 테이블에 대한 Query 문을 작성하세요

DEPT SUM(SALARY) ------- ----------- Group A 40900 Group B 134600

Group A : 부서번호 10, 20, 50Group B : 나머지 전부 (60 ~ 110, Null)SUM(SALARY) : 각 그룹별 Salary 컬럼의 값의 합계

답안>>SELECT (CASE WHEN DEPARTMENT_ID IN (10,20,50) THEN 'GROUPA' ELSE 'GROUPB' END) DEPT , SUM(SALARY)FROM EMPLOYEESGROUP BY (CASE WHEN DEPARTMENT_ID IN (10,20,50) THEN 'GROUPA' ELSE 'GROUPB' END)ORDER BY DEPT;

<< SUBQUERY>>

SELECT employee_id, last_name, job_id, salaryFROM employeesWHERE salary = ANY(SELECT salary FROM employees WHERE job_id = 'IT_PROG')AND job_id <> 'IT_PROG';

SELECT employee_id, last_name, job_id, salaryFROM employeesWHERE salary IN (SELECT salary FROM employees WHERE job_id = 'IT_PROG')AND job_id <> 'IT_PROG';

– <MULTIPLE COLUMN SUBQUERY>의 경우에는 쌍비교 방식 (PAIRWISE COMPARISON) 으로 데이터를 비교한다.– 두 개이상의 컬럼의 값을 하나로 묶어서 비교하는 방식

```
SELECT first_name, department_id, salaryFROM employeesWHERE (salary, department_id) IN(SELECT min(salary), department_idFROM employeesGROUP BY department_id)ORDER BY department_id;
```

– 비쌍비교방식 (NON PAIRWISE COMPARISON)– 각각의 컬럼을 개별적으로 비교하기 때문에 위의 결과와 다르게 나올 수 있다.

```
SELECT first_name, department_id, salaryFROM employeesWHERE salary IN (SELECT min(salary)FROM employeesGROUP BY department_id)AND department_id IN(SELECT department_idFROM employees)ORDER BY department_id;
```

– MULTIPLE COLUMN SUBQUERY의 경우에는 쌍 비교방식 (PAIRWISE COMPARISON) 을 사용한다.

– NOT IN 연산자를 사용할 경우 SUBQUERY의 결과에 NULL이 포함되어 있으면 MAIN QUERY의 결과도 NULL이 나오게 된다.
– 매니저 역할을 하고 있지 않는 사원조회하는 내용이지만 실제 조회는 안됨

```
SELECT emp.last_nameFROM employees empWHERE emp.employee_id NOT IN(SELECT mgr.manager_idFROM employees mgr);
```

```
SELECT emp.last_nameFROM employees empWHERE emp.employee_id != ALL(SELECT mgr.manager_idFROM employees mgr);
```

– 아래와 같은 QUERY를 사용하면 manager가 아닌 사원 조회 가능

SELECT emp.last_nameFROM employees empWHERE emp.employee_id NOT IN(SELECT mgr.manager_idFROM employees mgr WHERE mgr.manager_id is not null); (O)
– IN을 사용하는 경우에는 SUBQUERY에 NULL이 포함이 되어 있더라도 상관없다.
– 매니저 역할을 하고 있는 사원조회하는 내용이고 실제 조회 된다.
SELECT emp.last_nameFROM employees empWHERE emp.employee_id IN(SELECT mgr.manager_idFROM employees mgr); (O)
SELECT emp.last_nameFROM employees empWHERE emp.employee_id = ANY(SELECT mgr.manager_idFROM employees mgr); (O)
SELECT MAX(AVG(SALARY)) FROM EMPLOYEESGROUP BY DEPARTMENT_ID;
SELECT DEPARTMENT_ID,MAX(AVG(SALARY)) FROM EMPLOYEESGROUP BY DEPARTMENT_ID; (X)

SELECT DEPARTMENT_ID, AVG(SALARY) FROM EMPLOYEESHAVING AVG(SALARY) = (SELECT MAX(AVG(SALARY)) FROM EMPLOYEES GROUP BY DEPARTMENT_ID)
GROUP BY DEPARTMENT_ID; (O)

– IN LINE VIEW사원이 자신이 속해있는 부서의 평균급여보다 많은 급여를 받고 있는 사원 조회
SELECT E.EMPLOYEE_ID, E.LAST_NAME, E.SALARY, E.DEPARTMENT_ID, D.DEPTID, D.AVGSALFROM (SELECT DEPARTMENT_ID DEPTID, AVG(SALARY) AVGSAL FROM EMPLOYEES GROUP BY DEPARTMENT_ID) D, EMPLOYEES E WHERE E.DEPARTMENT_ID=D.DEPTID AND E.SALARY > D.AVGSAL;

<< 집합연산자 >>– SET OPERATOR : 집합 연산자-- 합집합, 교집합, 차집합 연산들이 SELECT문장의 결과에 대해 사용 가능.

```
-- 사원들의 과거 이력(부서나 직책의 변동) 정보 조회SELECT * FROM
JOB_HISTORY WHERE EMPLOYEE_ID=176;
-- 현재 근무정보 조회SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID
FROM EMPLOYEES WHERE EMPLOYEE_ID=176;

-- UNION ALL-- 합집합 {1,2,3} UNION ALL {1,2} => {1,2,3,1,2}SELECT
EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY WHERE
EMPLOYEE_ID=176UNION ALLSELECT
EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHERE
EMPLOYEE_ID=176;



-- UNION-- 합집합 {1,2,3} UNION {1,2} => {1,2,3}
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY
WHERE EMPLOYEE_ID=176UNION
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHERE EMPLOYEE_ID=176;

-- INTERSECT-- 교집합 {1,2,3} INTERSECT {1,2} ={1,2}
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY
WHEREEMPLOYEE_ID=176INTERSECT
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176;

-- MINUS (차집합)-- {1,2,3} MINUS {1,2} ={3}
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY
WHEREEMPLOYEE_ID=176MINUS
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176;

-- 일반적으로는 위에 있는 집합연산부터 실행
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY
WHEREEMPLOYEE_ID=176UNION ALL
```

```
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176MINUS
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176;
```

```
-- 괄호안에 있는 내용이 먼저 실행된다.
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY
WHEREEMPLOYEE_ID=176UNION ALL
(SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176MINUS
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176);
```

집합연산이 실행되려면 비교되는 SELECT문의 COLUMN 갯수가 일치해야 하고 서로 매치되는 COLUMN간의 데이터유형이 같아야 한다.하지만 매치되는 COLUMN의 이름은 달라도 상관없다.

```
– 집합연산시 가상의 컬럼을 사용하는 경우
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID,START_DATE,END_DATE
FROM JOB_HISTORY WHERE EMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID, JOB_ID, DEPARTMENT_ID, TO_DATE(NULL),
TO_DATE(NULL)FROM EMPLOYEES WHERE EMPLOYEE_ID=176;
```

```
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID,START_DATE,END_DATE
FROM JOB_HISTORY WHERE EMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,'FAKE',0,TO_DATE(NULL),TO_DATE(NULL)FROM
EMPLOYEES WHERE EMPLOYEE_ID=176;
```

```
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID,START_DATE,END_DATE
FROM JOB_HISTORY WHERE EMPLOYEE_ID=176UNION ALL
```

SELECT EMPLOYEE_ID, TO_CHAR(NULL), TO_NUMBER(NULL), TO_DATE(NULL),TO_DATE(NULL)FROM EMPLOYEES WHERE EMPLOYEE_ID=176;

SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID,START_DATE,END_DATE FROM JOB_HISTORY WHERE EMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,NULL,NULL,NULL,NULLFROM EMPLOYEES WHERE EMPLOYEE_ID=176; (O)

-- UNION, INTERSECT, MINUS 3개의 집합연산자의 경우는 연산 결과에 대해 자동 정렬이 발생– 자동정렬이 첫번째 컬럼을 기준으로 자동 정렬(ASC, 오름차순)을 한다.– UNION ALL을 사용하는 경우에는 자동 정렬을 되지 않는다.– UNION ALL의 결과에 대해 정렬을 하려면 ORDER BY절을 사용해야 한다.– ORDER BY 절은 집합연산자의 결과에 사용하는 경우 마지막 SELECT문장에서 사용해야 한다.– ORDER BY 절은 집합연산자의 결과에 사용하는 경우 중간에 나오는 SELECT문장에서는 사용불가
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176ORDER BY 1;

-- EMPLOYEE_ID컬럼을 기준으로 전체 결과에 대한 정렬
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY WHEREEMPLOYEE_ID=176ORDER BY 1 (X)UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176ORDER BY 1 (X)UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176;ORDER BY 1 (O)

SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY
WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES
WHEREEMPLOYEE_ID=176ORDER BY EMPLOYEE_ID; (O)

– 집합연산의 결과에 머리글은 첫번째 SELECT문의 컬럼이름이나 ALIAS가 나온다.

```
SELECT EMPLOYEE_ID EMPID1,JOB_ID,DEPARTMENT_ID FROM
JOB_HISTORY WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID EMPID2,JOB_ID,DEPARTMENT_ID FROM
EMPLOYEES WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID EMPID3,JOB_ID,DEPARTMENT_ID FROM
EMPLOYEES WHEREEMPLOYEE_ID=176;
```

– 컬럼에 ALIAS가 설정되면 컬럼의 이름을 ORDER BY절에 직접 사용할 수 없게 된다.

```
SELECT EMPLOYEE_ID EMPID1,JOB_ID,DEPARTMENT_ID FROM
JOB_HISTORY WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID EMPID2,JOB_ID,DEPARTMENT_ID FROM
EMPLOYEES WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID EMPID3,JOB_ID,DEPARTMENT_ID FROM
EMPLOYEES WHEREEMPLOYEE_ID=176ORDER BY EMPLOYEE_ID; (X)
```

SELECT EMPLOYEE_ID EMPID1,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID EMPID2,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID EMPID3,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176ORDER BY EMPID1; (X)
SELECT EMPLOYEE_ID EMPID1,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176ORDER BY 1; (O)

– ORDER BY절에 컬럼이름을 사용하려면 ALIAS를 따로 주지 않고 사용
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM JOB_HISTORY WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176UNION ALL
SELECT EMPLOYEE_ID,JOB_ID,DEPARTMENT_ID FROM EMPLOYEES WHEREEMPLOYEE_ID=176ORDER BY EMPLOYEE_ID; (O)

11장 DDL (DATA DEFINITION LANGUAGE) : SQL명령의 한 종류-
데이터베이스 객체 (DATABASE Object) 생성 관리에 사용되는 SQL명령1. CREATE, ALTER, DROP과 같은 DDL SQL명령을 이용해서 데이터베이스 객체 생성, 관리2. 데이터베이스 객체는 여러가지 종류가 있는데 그중에 대표적인 것은 테이블이다.3. 데이터베이스 객체는 특정 데이터베이스 USER(예.user01.. userxx USER)가 소유(OWNER)하게 된다. 예를 들면 USER10 계정으로 DB에 LOGIN하고 테이블을 생성하면 해당 테이블은 USER10 계정이 소유하는 테이블이 된다. 소유한다는 것은 해당 테이블에 대한 모든 권한을 가지게 된다는 의미를 가진다. 데이터베이스 객체는 특정 USER의 SCHEMA안에 생성이 된다 SCHEMA(스키마)는 특정 USER가 소유하고 있는 데이터베이스 객체들의 집합을 가리킨다.

SCHEMA (스키마) : SCHEMA는 특정 USER가 소유하고 있는 객체들의 집합USER가 소유하고 있다는 것은 해당 USER가 DDL 명령으로 직접 생성을 했다는 것이다.객체의 소유자가 되면 해당 객체에 대한 모든 작업의 권한을 가지게 된다.객체 관련된 소유자가 아닌 다른 USER들은 해당 객체에 대해서는 어떠한 작업도 할 수 없다.객체를 소유하고 있는 USER가 다른 USER에게 권한(조회,데이터 입력 등)을 별도로 주어야만 해당 USER는 작업이 가능해진다. 권한은 뺏을 수도 있다 이러한 권한을 관리할 때 사용하는 명령이 DCL(DATA CONTROL LANGUAGE)이다.

VARCHAR2 TYPE과 CHAR TYPE의 차이점DROP TABLE TEST;
CREATE TABLE TEST(ID VARCHAR2(10));
– DML(DATA MANIPULATION LANGUAGE)INSERT INTO TEST(ID) VALUES('ABC'); => 실제로 ID COLUMN에는 3자리 문자 데이터 저장

DROP TABLE TEST2;CREATE TABLE TEST2 (ID2 CHAR(10));
INSERT INTO TEST2(ID2) VALUES('ABC'); => 실제로 ID COLUMN에는 10자리 문자 데이터 저장10자리의 공간 안에 3자리에만 데이터가 들어가고 나머지 7자리는 빈공간(SPACE)으로 채워지게 된다.

CHAR만 잡았을 경우 SIZE를 따로 안 잡아도 된다. 실제로 1개의 문자만 들어올 수 있게 된다.DROP TABLE TEST3;CREATE TABLE TEST3 (ID2 CHAR); => (ID2 CHAR(1)) 와 같은 의미가 된다.VARCHAR2 데이터 유형의 경우는 반드시 SIZE를 잡아 주어야 한다.
CREATE TABLE TEST(ID VARCHAR2); (X)NUMBER[(p,s)] : 숫자 데이터, P (Precision) : 총 자리수, S (scale) : 소수점 아래 자릿수
DROP TABLE TEST;
CREATE TABLE TEST(ID NUMBER(5,2));
INSERT INTO TEST VALUES(123.56); (O)
INSERT INTO TEST VALUES(123.567); (O) => 123.57
INSERT INTO TEST VALUES(1234.56); (X)SELECT * FROM TEST;

Datetime 데이터 타입(데이터 유형)
1.TIMESTAMP (TS) : 날짜 및 시,분,초, Fractional Second (초 단위 이하)
2.TIMESTAMP WITH TIME ZONE (TSTZ) : TIMESTAMP + Time Zone 정보

3.TIMESTAMP WITH LOCAL TIME ZONE (TSLTZ) : TIMESTAMP + DB Time Zone 기준으로 자동 변환

<<예제>>CREATE TABLE TIME_TABLE (TIME1 TIMESTAMP, TIME2 TIMESTAMP WITH TIME ZONE, TIME3 TIMESTAMP WITH LOCAL TIME ZONE);

– DB 사용자가 있는 지역의 현재 날짜 시간

SELECT CURRENT_TIMESTAMP FROM DUAL;

INSERT INTO TIME_TABLE (TIME1,TIME2,TIME3)VALUES (CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);

SELECT TIME1 FROM TIME_TABLE; SELECT TIME2 FROM TIME_TABLE; SELECT TIME3 FROM TIME_TABLE;

ALTER SESSION SET TIME_ZONE = '-4:00'; – 현재 사용자가 있는 지역의 시간대를 임시로 변경

SELECT CURRENT_TIMESTAMP FROM DUAL;

SELECT TIME1 FROM TIME_TABLE;

SELECT TIME2 FROM TIME_TABLE;

SELECT TIME3 FROM TIME_TABLE;

-- TIMESTAMP WITH LOCAL TIME ZONE 데이타 타입으로 잡힌 컬럼에 날짜와 시간은 데이터를 조회하는 지역의 시간대에 따라 저장되어 있는 날짜와 시간이 같이 자동 변환해서 조회

INTERVAL YEAR TO MONTH : 연, 월 단위의 기간INTERVAL DAY TO SECOND : 일, 시, 분, 초 단위의 기간

<<예제>>ALTER SESSION SET TIME_ZONE = LOCAL;

SELECT CURRENT_TIMESTAMP FROM DUAL;

CREATE TABLE time_test2(ym INTERVAL YEAR TO MONTH, ds INTERVAL DAY TO SECOND);

INSERT INTO time_test2(YM, DS)VALUES(INTERVAL '10-5' YEAR TO MONTH, INTERVAL '10 05:22:15' DAY TO SECOND);

SELECT CURRENT_TIMESTAMP+YM FROM time_test2; => 현재 날짜 기준으로 10년 5개월 후의 날짜

<< 날짜 데이터 입력 (DEFAULT이용)>>

ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';

CREATE TABLE hire_dates (id NUMBER(8), hire_date DATE DEFAULT CURRENT_DATE);
SELECT CURRENT_DATE FROM DUAL;
INSERT INTO HIRE_DATESVALUES(1234,DEFAULT);
INSERT INTO HIRE_DATESVALUES(1234,'1988-02-23'); (O)
INSERT INTO HIRE_DATESVALUES(1234,'02-23-2011'); (X)
INSERT INTO HIRE_DATES
VALUES(1234,TO_DATE('02-23-2011','MM-DD-YYYY')); (O)

<< DEFAULT>>
CREATE TABLE T1 (ID NUMBER DEFAULT 10);
INSERT INTO T1(ID) VALUES(10); (O)
INSERT INTO T1(ID) VALUES(DEFAULT); (O)

<<제약조건 (CONSTRAINT)>>제약조건은 컬럼에 셋업하는 것.

\1. PRIMARY KEY COLUMN => 테이블 안에 있는 컬럼들 중에 대표가 되는 컬럼이다.=> PRIMARY KEY COLUMN이 되기 위해서는 조건(규칙)이 있는데첫번째 조건은 COLUMN안에 들어가는 데이타가 중복이 되면 안된다.(UNIQUE해야 한다)두번째 조건은 NULL이 들어가면 안된다.(COMMISSION_PCT 컬럼의 경우는 PRIMARY KEY COLUMN이 될 수 없다)하나 이상의 컬럼이 하나의 PRIMARY KEY로 지정 가능하나의 테이블에 하나의 PRIMARY KEY제약조건만 존재 가능

-- 컬럼 단위(column level)로 제약조건 생성
DROP TABLE DEPT_TEST;
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) CONSTRAINT DEPT_TEST_DEPTNO_PK PRIMARY KEY,DNAME VARCHAR2(14),LOC VARCHAR2(13));

-- 테이블 단위(table level)로 생성DROP TABLE DEPT_TEST;

```
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13),CONSTRAINT
DEPT_TEST_DEPTNO_PK PRIMARY KEY(DEPTNO));
-- 컬럼 단위(column level)로 생성
DROP TABLE DEPT_TEST;CREATE TABLE DEPT_TEST(DEPTNO
NUMBER(2) PRIMARY KEY, -- 제약조건의 이름이 SYS_Cn으로 자동
생성DNAME VARCHAR2(14),LOC VARCHAR2(13));



-- 제약조건의 정보를 조회 (DATA DICTIONARY VIEW 이용)
COL TABLE_NAME FORMAT A15
COL COLUMN_NAME FORMAT A15
COL CONSTRAINT_NAME FORMAT A20
SELECT TABLE_NAME,COLUMN_NAME,CONSTRAINT_NAMEFROM
USER_CONS_COLUMNS -- DATA DICTIONARY VIEWWHERE
TABLE_NAME = 'DEPT_TEST';
DELETE FROM DEPT_TEST;
INSERT INTO DEPT_TEST VALUES(10,'FIN','SEOUL'); (O)
SELECT * FROM DEPT_TEST;INSERT INTO DEPT_TEST
VALUES(10,'MARK','SEOUL'); (X)
INSERT INTO DEPT_TEST VALUES(NULL,'MARK','SEOUL'); (X)
INSERT INTO DEPT_TEST VALUES(20,'MARK','SEOUL'); (O)

<< 두 개 이상의 컬럼을 묶어서 하나의 PRIMARY KEY로 만드는 예>>
DROP TABLE DEPT_TEST2
CREATE TABLE DEPT_TEST2(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13),CONSTRAINT
DEPT_TEST_DEPTNO_DNAME_PK PRIMARY KEY(DEPTNO,DNAME));
INSERT INTO DEPT_TEST2 VALUES(10,'A','SEOUL'); (O)
INSERT INTO DEPT_TEST2 VALUES(10,'B','SEOUL'); (O)
SELECT * FROM DEPT_TEST2;
INSERT INTO DEPT_TEST2 VALUES(10,'A','BUSAN''); (X)
```

INSERT INTO DEPT_TEST2 VALUES(10,NULL,'SEOUL'); (X)

-- DEPTNO 컬럼을 PRIMARY KEY로 추가 지정하는 작업
DROP TABLE DEPT_TEST;
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13));

– 기존의 DEPTNO 컬럼에 NULL인 상태가 있다거나 중복되는 데이터가 있다면 아래의 작업이 실패
– 아래의 명령이 성공하려면 기존의 데이터 중에 NULL인 상태나 중복되는 데이터가 없어야 한다.
ALTER TABLE DEPT_TESTADD CONSTRAINT DEPT_TEST_PK_DEPT
PRIMARY KEY (DEPTNO);

– PRIMARY KEY 제약조건을 비활성화 (DEPTNO컬럼에는 NULL이나 중복된값 입력이 가능)
ALTER TABLE DEPT_TEST DISABLE CONSTRAINT DEPT_TEST_PK_DEPT;

– PRIMARY KEY 제약조건을 활성화
– 기존의 DEPTNO 컬럼에 NULL인 상태가 있다거나 중복되는 데이터가 있다면 ENABLE 작업이 실패
ALTER TABLE DEPT_TEST ENABLE CONSTRAINT DEPT_TEST_PK_DEPT;

-- PRIMARY KEY 제약조건을 삭제 (DEPTNO컬럼에는 NULL이나 중복된값 입력이 가능)
ALTER TABLE DEPT_TESTDROP CONSTRAINT DEPT_TEST_PK_DEPT;

\2. UNIQUE 제약조건 : 중복되는 데이터 입력을 금지 (보조키값= (#)), NULL입력 가능

-- 컬럼 단위(COLUMN level)로 생성
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) CONSTRAINT
DEPT_TEST_DEPTNO_UK UNIQUE,DNAME VARCHAR2(14) CONSTRAINT
DEPT_TEST_DNAME_UK UNIQUE,LOC VARCHAR2(13));

-- 테이블 단위(table level)로 생성
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13),CONSTRAINT
DEPT_TEST_DEPTNO_UK UNIQUE(DEPTNO),CONSTRAINT
DEPT_TEST_DNAME_UK UNIQUE(DNAME));

CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) UNIQUE,
-- 제약조건의 이름이 SYS_Cn으로 자동 생성
DNAME VARCHAR2(14),LOC VARCHAR2(13));
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(10,'IT','SEOUL'); (O)
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(10,'ACCT','BUSAN'); -- (X)
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(NULL,'CORP','BUSAN'); -- (O)

\3. NOT NULL 제약조건 : NULL의 입력을 금지

-- 컬럼 단위(COLUMN level)로 생성
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) CONSTRAINT
DEPT_TEST_DEPTNO_NN NOT NULL,DNAME VARCHAR2(14),LOC
VARCHAR2(13));

CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) NOT NULL,
-- 제약조건의 이름이 SYS_Cn으로 자동 생성DNAME VARCHAR2(14),LOC
VARCHAR2(13));
-- 테이블 단위(table level)로 생성할 수 없다DROP TABLE DEPT_TEST;
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13),CONSTRAINT
DEPT_TEST_DEPTNO_NN NOT NULL(DEPTNO)); (X)

INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(NULL,'ACCT','BUSAN'); – (X)
\4. CHECK 제약조건-- 컬럼 단위(column level)로 생성
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) CONSTRAINT
DEPT_TEST_DEPTNO_CK CHECK(DEPTNO > 10), DNAME
VARCHAR2(14),LOC VARCHAR2(13)); (O)
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2) CONSTRAINT
DEPT_TEST_DEPTNO_CK CHECK(DEPTNO > 10 AND DNAME > 'A'), (X)
DNAME VARCHAR2(14),LOC VARCHAR2(13));

-- 테이블 단위(table level)로 생성할 수 있다.
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13),CONSTRAINT
DEPT_TEST_DEPTNO_CK CHECK(DEPTNO > 10)); (O)
CREATE TABLE DEPT_TEST(DEPTNO NUMBER(2),DNAME
VARCHAR2(14),LOC VARCHAR2(13),CONSTRAINT
DEPT_TEST_DEPTNO_CK CHECK(DEPTNO > 10 AND DNAME > 'A')); (O)
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(20,'ACCT','BUSAN'); (O)
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(30,'A','BUSAN'); (X)
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(5,'ACCT','BUSAN'); (X)

\5. FOREIGN KEY 제약조건: CHILD TABLE(DETAIL TABLE)쪽에 만들어지는 제약조건.DEPTNO의 경우는 어떤 사원이 어느 부서에 소속이 되어 있는지에 대한 정보를 보여주는 컬럼에 설정:FOREIGN KEY 제약조건을 만들려면 반드시 PARENT TABLE(DEPT_TEST)에 PRIMARY KEY제약조건이 설정되어 있어야 한다. FOREIGN KEY제약조건 생성시에 PRIMARY KEY 조건이 걸려있는 컬럼정보가 필요.

DROP TABLE DEPT_TEST;

```
CREATE TABLE DEPT_TEST -- PARENT, MASTER TABLE(DEPTNO
NUMBER(2) CONSTRAINT DEPT_TEST_DEPTNO_PK PRIMARY
KEY,DNAME VARCHAR2(14),LOC VARCHAR2(13));
-- 컬럼 단위(column level)로 생성
CREATE TABLE EMP_TEST -- CHILD, DETAIL TABLE(EMPNO
NUMBER(2),ENAME VARCHAR2(10),DEPTNO NUMBER(2) CONSTRAINT
EMP_9TEST_DEPTNO_FK REFERENCESDEPT_TEST(DEPTNO));

-- -- 테이블 단위(table level)로 생성
CREATE TABLE EMP_TEST(EMPNO NUMBER(2),ENAME
VARCHAR2(10),DEPTNO NUMBER(2),CONSTRAINT
EMP_TEST_DEPTNO_FK FOREIGN KEY(DEPTNO)REFERENCES
DEPT_TEST(DEPTNO));
```

– FOREIGN KEY 컬럼에는 중복되는 데이타가 들어 올 수 있고 NULL인 상태도 될 수 있다.

```
DELETE FROM DEPT_TEST;
DELETE FROM EMP_TEST;
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(10,'ACCT','BUSAN');
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(20,'BCCT','SEOUL');
INSERT INTO DEPT_TEST (DEPTNO, DNAME, LOC) VALUES
(30,'FUUU','SEOUL');
INSERT INTO EMP_TEST (EMPNO,ENAME,DEPTNO) VALUES(90,'A',10);  (O)
INSERT INTO EMP_TEST (EMPNO,ENAME,DEPTNO) VALUES(91,'B',NULL);
(O)
INSERT INTO EMP_TEST (EMPNO,ENAME,DEPTNO) VALUES(92,'C',20); (O)
INSERT INTO EMP_TEST (EMPNO,ENAME,DEPTNO) VALUES(93,'U',20); (O)
```

-- EMP_TEST 테이블(FOREIGN KEY가 생성되어 있는 테이블)에 FOREIGN KEY 컬럼(DEPTNO)에 DEPT_TEST 테이블에 없는 부서를 입력하려고 하는 경우 데이타의 입력 불가능.
INSERT INTO EMP_TEST (EMPNO,ENAME,DEPTNO) VALUES(94,'F',50); (X)
-- FOREIGN KEY 컬럼이 생성이 되면 FOREIGN KEY COLUMN과 연결되어 있는부서테이블의 부서정보를 삭제할 때 삭제하려고 하는 부서와 관련된 사원정보가 사원테이블에 남아 있는 경우에 부서테이블의 부서정보를 삭제하지 못하게 된다.
DELETE FROM DEPT_TEST WHERE DEPTNO=10; (X)
DELETE FROM DEPT_TEST WHERE DEPTNO=20; (X)
DELETE FROM DEPT_TEST WHERE DEPTNO=30; (O)

\- FOREIGN KEY OPTION1. ON DELETE SET NULL: FOREIGN KEY 컬럼이 생성이 되면 FOREIGN KEY COLUMN과 연결되어 있는 부서테이블의 부서정보를 삭제할 때 삭제하려고 하는 부서의 관련된 사원정보가 사원 테이블에 남아 있는 경우에 부서테이블의 부서정보는 삭제되면서 사원 테이블의 부서번호 데이터를 NULL로 자동 변경.
DROP TABLE EMP_TEST;
CREATE TABLE EMP_TEST(EMPNO NUMBER(2),ENAME VARCHAR2(10),DEPTNO NUMBER(2),CONSTRAINT EMP_TEST_DEPTNO_FK FOREIGN KEY(DEPTNO)REFERENCES DEPT_TEST(DEPTNO) **ON DELETE SET NULL**);

\2. ON DELETE CASCADE: FOREIGN KEY 컬럼이 생성이 되면 FOREIGN KEY COLUMN과 연결되어 있는 부서테이블의 부서정보를 삭제할 때 삭제하려고 하는 부서의 관련된 사원정보가 사원 테이블에 남아 있는 경우에 사원 테이블의 사원 데이터를 같이 자동삭제.
DELETE FROM DEPT_TEST WHERE DEPTNO=10; (O)=> 10번부서와 관련된 사원 데이타들이 같이 삭제됨
DROP TABLE EMP_TEST;
CREATE TABLE EMP_TEST(EMPNO NUMBER(2),ENAME VARCHAR2(10),DEPTNO NUMBER(2),CONSTRAINT EMP_TEST_DEPTNO_FK FOREIGN KEY(DEPTNO)REFERENCES DEPT_TEST(DEPTNO) **ON DELETE CASCADE**);

-- CTAS (CREATE TABLE AS SELECT)CREATE TABLE dept60AS SELECT employee_id, last_name, salary, hire_date, DEPARTMENT_IDFROM EMPLOYEES WHERE DEPARTMENT_ID=60;
DROP TABLE DEPT60;
CREATE TABLE dept60(EMPID,ENAME,SAL,HIREDATE,DEPTID)AS SELECT employee_id , last_name, salary, hire_date, DEPARTMENT_IDFROM EMPLOYEES WHERE DEPARTMENT_ID=60;SELECT * FROM DEPT60;
DROP TABLE DEPT60;CREATE TABLE dept60AS SELECT employee_id EMPID, last_name ENAME, salary SAL, hire_date, DEPARTMENT_IDFROM EMPLOYEES WHERE DEPARTMENT_ID=60;SELECT * FROM DEPT60;
DROP TABLE DEPT60;CREATE TABLE dept60AS SELECT employee_id , last_name, salary, hire_date, DEPARTMENT_IDFROM EMPLOYEES WHERE 3=2;– WHERE절에는 거짓인 조건을 입력하면 데이터는 COPY하지 않고 테이블의 구조만 COPY하게 된다.

-- ALTER TABLECREATE TABLE TEST_T7 (ID NUMBER);DESC TEST_T7

– NAME, EMAIL 컬럼을 추가
ALTER TABLE TEST_T7 ADD (NAME VARCHAR2(20),EMAIL VARCHAR2(10));
DESC TEST_T7

– 컬럼의 데이터 유형이나 최대 크기 혹은 DEFAULT값 변경
ALTER TABLE TEST_T7 MODIFY (NAME CHAR(30) DEFAULT 'KIM');-- 수정가능한 것은 COLUMN의 DATA TYPE과 SIZE, DEFAULT값, 제약조건도 추가 가능)– SIZE 조정시 만약에 해당 컬럼에 데이터가 이미 존재하는 경우에는 기존의 데이터보다 작게 조정할 수 없다.

– NAME COLUMN 삭제
DESC TEST_T7
ALTER TABLE TEST_T7
DROP COLUMN NAME;
-- DROP (COL1,COL2)DESC TEST_T7

– COLUMN의 이름 변경
ALTER TABLE TEST_T7
RENAME COLUMN EMAIL TO EMAIL2;
DESC TEST_T7

– TABLE의 이름 변경
RENAME TEST_T7 TO TEST_T8;

-- INSERT/UPDATE/DELETE를 할 수 없는 TABLE, SELECT만 할 수 있는 TABLEALTER TABLE TEST_T8 READ ONLY;

--SELECT/INSERT/UPDATE/DELETE를 할 수 있는 TABLE로 변경ALTER TABLE TEST_T8 READ WRITE;

-- SET UNUSED – 컬럼이 삭제가 된 것처럼 만드는 작업, 실제로 컬럼이 삭제되는 것은 아니다.
CREATE TABLE TEST_U (ID NUMBER,NAME VARCHAR2(20));DESC TEST_U
ALTER TABLE TEST_U SET UNUSED COLUMN NAME;– 컬럼을 DROP을 하게 되면 시간도 오래 걸리고 DROP을 하는 동안 해당 테이블에는 LOCK이 걸려서– 다른 사용자들이 데이터를 변경하는 작업을 하지못하게 된다.-- NAME COLUMN을 더이상 사용하지 않겠다 (DROP(삭제)된 것처럼 만든다.) – 실제로 삭제된 것이 아니다– SELECT할 때나 DESC로 구조를 조회할시에 해당 컬럼이 조회되지 않게 된다. – UNUSED로 MARKING을 하는 동안에도 해당 테이블에는 LOCK이 걸려서– 다른 사용자들이 데이터를 변경하는 작업을 하지 못하게 된다. DROP할때보다는 오래 걸리지 않는다.

-- UNUSED로 셋업된 COLUMN들을 모두 한꺼번에 실제로 DROP된다.ALTER TABLE TEST_U DROP UNUSED COLUMNS;

– 한 번 UNUSED로 MARKING된 컬럼을 다시 사용할 수 없다.ALTER TABLE TEST_U SET USED COLUMN NAME; (X)

– 그러나 ALTER TABLE TEST_U SET UNUSED COLUMN NAME ONLINE; 로 실행하면 – UNUSED로 MARKING을 하는 동안에도 해당 테이블에는 대해서– 다른 사용자들이 데이터를 변경하는 작업이 가능하지만 MARKING하는 작업이 오래 걸릴 수 있다.

-- DROP TABLECREATE TABLE TEST_F (ID NUMBER,NAME VARCHAR2(20));

DROP TABLE TEST_F;

– RECYCLE BIN(휴지통)에 잠시 저장 가능

SELECT * FROM USER_RECYCLEBIN; – 휴지통안에 DROP된 테이블 정보 조회

PURGE RECYCLEBIN; – 휴지통 비우기

-- DROP되기 전의 상태로 다시 복원시키는 명령

FLASHBACK TABLE TEST_F TO BEFORE DROP;

DROP TABLE TEST_U PURGE; -- 휴지통에 들어가지 않고 바로 삭제.

\- DELETE와 TRUNCATE의 공통점은 테이블안에 있는 데이터를 삭제하는 용도로 사용 가능

– DELETE와 TRUNCATE의 차이점 DELETE 명령(DML)으로 RECORD 삭제하는 경우는 ROLLBACK(취소)할 수 있다 TRUNCATE 명령어(DDL)으로 RECORD 삭제하는 경우는 자동 COMMIT이 발생해서 ROLLBACK(취소)할 수 없다.DELETE 명령으로 RECORD 삭제하는 경우는 WHERE절 사용 가능 TRUNCATE 명령으로 RECORD 삭제하는 경우는 WHERE절 사용 불가능 TRUNCATE 명령으로 RECORD 삭제하는 경우는 무조건 전체 데이터 삭제만 가능

전체 테이블 데이타를 같은 조건에서 삭제작업을 진행시 TRUNCATE 명령으로 RECORD 삭제하는 경우는 훨씬 속도가 빠르다.

-- TRANSACTION이란하나이상의 DML 명령(INSERT/UPDATE/DELETE)들이 모여서 만들어지는 작업그룹작업그룹 단위로 COMMIT작업(작업내용을 DB안에 저장완료)과 ROLLBACK작업(작업 취소) 진행 가능하지만 한 번 COMMIT한 작업은 ROLLBACK할 수 없고 한번 ROLLBACK한 작업은 COMMIT할 수 없다.
INSERT INTO TEST_USER10 VALUES('GGG','AB'); -- TRANSACTION 자동 생성되고 시작
INSERT INTO TEST_USER10 VALUES('GGG','AB');
UPDATE DELETE COMMIT; -- 위에 실행된 DML명령의 결과들이 데이터베이스에 저장되고 TRANSACTION이 종료
INSERT INTO TEST_USER10 VALUES('YYY','XX'); -- TRANSACTION 자동 생성되고 시작INSERT INTO TEST_USER10 VALUES('YYY','XX');
ROLLBACK;
-- 위의 2개의 INSERT작업 취소되고 TRANSACTION이 종료
INSERT INTO TEST_USER10 VALUES('YYY','XX'); -- TRANSACTION 자동 생성되고 시작INSERT INTO TEST_USER10 VALUES('YYY','XX');UPDATE DELETE COMMIT; -- 데이터베이스에 저장되고 TRANSACTION이 종료ROLLBACK; -- 이미 COMMIT이 된 내용은 ROLLBACK할 수 없다

INSERT INTO TEST_USER10 VALUES('YYY','XX'); -- TRANSACTION 자동 생성되고 시작
INSERT INTO TEST_USER10 VALUES('YYY','XX');
UPDATE DELETE ROLLBACK; 위의 작업들이 취소되고 TRANSACTION이 종료COMMIT;
-- 이미 위의 TRANSACTION내용은 취소된 상태이기 때문에 COMMIT은 실행되지 않는다.

CREATE TABLE TEST_USER10 (ID VARCHAR2(10), NAME VARCHAR2(10));
-- DDL SQL명령어들은 각 문장 단위로 자동으로 COMMIT이 진행.
ROLLBACK 불가능
– DDL SQL명령은 명령문 단위로 명령문이 시작시 TRANSACTION이 만들어지게 된다.

– DDL명령 실행이 정상적으로 종료되면 COMMIT이 자동 실행되면서 TRANSACTION 자동 종료

INSERT INTO TEST_USER10 VALUES('YYY','XX');
-- TRANSACTION 자동 생성되고 시작
INSERT INTO TEST_USER10 VALUES('YYY','XX');
UPDATE DELETE CREATE TABLE YYY (ID NUMBER); – DDL명령을 실행하면 자동 COMMIT이 실행된다.

– 위에서 실행된 DML명령들에 대한 TRANSACTION도 자동 COMMIT– 위에서 실행된 DML명령들에 대한 TRANSACTION 종료

-- 현재 세션이 정상종료(로그아웃)시에는 COMMIT을 하지 않은 상태의 작업중이었던 TRANSACTION은 자동 COMMIT이 진행된다., TRANSACTION 종료하고 작업 저장

-- 현재 세션이 비정상 종료시에는 COMMIT을 하지 않은 상태의 작업중이었던 TRANSACTION은 자동 ROLLBACK, TRANSACTION 종료하고 작업 취소
명령문 레벨 롤백 INSERT UPDATE DELETE – ERROR– DELETE 명령문만 롤백이 된다.COMMIT; – INSERT와 UPDATE는 저장된다.

LOCK : DML작업을 진행하면 작업이 진행중인 RECORD(ROW)에 대해 LOCK이 걸리게 된다. LOCK이 걸리면 다른 사용자들은 LOCK이 걸린 RECORD에 대한 조회는 가능하지만 또다른 DML작업의 진행이 불가능하다. LOCK을 풀어주는 방법은 COMMIT이나 ROLLBACK명령을 실행하는 것이다.

--읽기 일관성 (READ CONSISTENCY)DML작업의 결과는 COMMIT이 되거나 ROLLBACK 된 이후의 작업결과가 다른 사용자들과 공유가 될 수 있다. DML작업을 진행하고 있는 중에는 작업결과가 다른 사용자들과 공유가 될 수 없다.

어떤 DB 사용자가 SELECT(데이터 조회) 작업을 시작하면 데이터를 조회하기 시작한 시점의 데이터를 기준으로 조회가 끝날때까지 보게된다.

DB USER #1 9:00 SELECT * FROM EMPLOYEES ;
-- 이 SELECT문장이 10분동안 진행된다고 가정 (조회종료시점 9:10)9:09 205번 사원 데이터를 조회
-- 이 때 205 사원에 대한 급여 데이터가 4000으로 보여야 한다.==> 9시에 시작한 QUERY(SELECT)는 QUERY가 끝날 때까지 9시 시점에 존재하는 데이터를 기준으로 조회를 하게 된다.9:11 SELECT * FROM EMPLOYEES;DB USER #2가 UPDATE 작업 결과를 조회할 수 있다. 5000)

DB USER #29:05 UPDATE EMPLOYEES SET ---- 변경 205번사원의 급여를 4000 => 5000 변경 9:06 SELECT (205번 사원 조회 하면, 5000으로 보인다)9:08 COMMIT;9:09 SELECT (205번 사원 조회 하면, 5000으로 보인다)
FOR UPDATE: 일반적으로 SELECT를 진행하는 경우에 SELECT하고 있는 RECORD에 LOCK이 걸리지는 않는다. SELECT 문에 FOR UPDATE절을 포함시키면 SELECT하는 RECORD에도 LOCK이 걸릴 수 있다.

-- SAVEPOINT : TRANSACTION안에서 ROLLBACK POINTER를 설정하는 작업INSERT INTO SAMPLE_TRAN --TRANSACTION 시작(EMPLOYEE_ID,LAST_NAME,SALARY,DEPARTMENT_ID)VALUES (120,'KIM',3000,30);
SAVEPOINT A; -- TRANSACTION이 종료되지 않는다, rollback pointer가 marking된다
INSERT INTO SAMPLE_TRANVALUES (121,'LEE',5000,30);
SAVEPOINT B; -- TRANSACTION이 종료되지 않는다, rollback pointer가 marking된다
INSERT INTO SAMPLE_TRAN(EMPLOYEE_ID,LAST_NAME,SALARY,DEPARTMENT_ID)VALUES (122,'PARK',7000,30);SAVEPOINT C; -- TRANSACTION이 종료되지 않는다, rollback pointer가 marking된다
INSERT INTO SAMPLE_TRAN(EMPLOYEE_ID,LAST_NAME,SALARY,DEPARTMENT_ID)VALUES (123,'OH',9000,30);

\1. ROLLBACK; – 위의 작업내용들이 모두 취소되고 SAVEPOINT들도 다 삭제

\2. ROLLBACK TO SAVEPOINT B; – savepoint b 아래의 작업내용들을 작업취소 savepoint b를 포함해서 아래의 SAVEPOINT정보도 삭제됨 savepoint b 위에 있는 내용들은 취소되지 않음 COMMIT; SAVEPOINT B 위의 작업들은 COMMIT이 완료되고 SAVEPOINT들은 다 삭제

\3. ROLLBACK TO SAVEPOINT A; – savepoint a 아래의 작업내용들을 취소, savepoint a 를 포함해서 아래의 SAVEPOINT정보도 삭제됨 savepoint a 위에 있는 내용들은 취소되지 않음 COMMIT; – savepoint a 위에 있는 첫번째 insert문은 commit이 된다.

\4. COMMIT; 위의 모든 작업들의 결과가 DB에 저장되고 SAVEPOINT들은 모두 삭제됨