1. Database 개요

1-1 데이터베이스(Database)란?

지속적으로 저장되는 연관된 정보의 모음. 특정 관심의 데이터를 수집하여 그 데이터의 성격에 맞도록 잘 설계하여 저장하고 관리함으로써 필요한 데이터를 효율적으로 사용할 수 있는 자원

1-2 DBMS(Database Management System)

- 데이터를 효율적으로 관리할 수 있는 시스템
- 데이터를 효율적으로 관리하기 위해서 데이터베이스에 추가, 삭제, 변경, 검색을 할 수 있는 기능 제공

1-3 관계형 데이터 베이스

- E.F. Codd 박사는 1970년에 데이터베이스 시스템을 위한 관계형 모델을 제안.
- 이것이 RDBMS(관계형 데이터베이스 관리 시스템)의 기초
- 관계형 모델의 구성

1) 데이터를 저장하는 객체 또는 관계의 모음
2) 관계에 적용되어 다른 관계 생성을 할 수 있도록 해주는 연산자 집합
3) 정확성과 일관성을 위한 데이터 무결성

- 관계형 데이터 베이스의 특징

1) **SQL**문을 실행하여 액세스하고 수정
2) 물리적인 포인터가 없는 테이블을 가고 있다.
3) 연산자 집합을 사용한다.

2. SQL

2-1. SQL(Standard Query Language)이란?

- **SQL**은 데이터베이스에서 데이터를 검색, 삽입, 갱신, 삭제할 수 있는 표준 언어
- SQL은 1970년대 IBM에 의해서 처음 공개된 이후 ANSI/ISO 표준으로 편입됨

2-2. SQL문의 종류

DDL(Data Definition Language) : 데이터와 그 구조를 정의

SQL문	내용
create	데이터베이스 객체를 생성
drop	데이터베이스 객체를 삭제
alter	기존에 존재하는 데이터베이스 객체를 다시 정의하는 역할

DML(Data Manipulation Language) : 데이터의 검색과 수정 등의 처리

SQL문	내용
insert	데이터베이스 객체에 데이터를 입력
delete	데이터베이스 객체에 데이터를 삭제
update	기존에 존재하는 데이터베이스 객체안의 데이터 수정
select	데이터베이스 객체로부터 데이터를 검색

DCL(Data Control Language) : 데이터베이스 사용자의 권한을 제어

SQL문	내용
grant	데이터베이스 객체에 권한을 부여
revoke	이미 부여된 데이터베이스 객체의 권한을 취소

TCL(Transaction Control Language) : 데이터베이스 트랜잭션 제어

SQL문	내용
commit	보류 중인 모든 변경 내용을 영구히 저장
rollback	저장점 표시자까지 롤백하는 데 사용
savepoint	보류 중인 데이터 변경 내용을 모두 버림

- SQL문의 작성

1) SQL 문은 대소문자를 구분하지 않음 2) SQL 문은 한 줄 또는 여러 줄에 입력할 수 있음 3) 키워드는 약어로 표기하거나 여러 줄에 걸쳐 입력할 수 없음 4) 절은 대개 별도의 줄에 입력함

5) 가독성을 높이기 위해 들여쓰기 사용
6) 키워드는 일반적으로 대문자로 입력하지만 테이블 이름, 열 이름 등의 다른 단어는 모두 소문자로 입력

2-3. SELECT 문

데이터베이스로부터 저장되어 있는 데이터를 검색하는데 사용
키워드 (SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY)
절(clause) : 키워드가 포함된 라인
문(statement) : 하나 이상의 절이 결합

1) 구문

```
select {*|[distinct] column|expression [alias],...}
from table;
```

SELECT	둘 이상의 열로 이루어진 리스트
*	모든 열을 선택
DISTINCT	중복 방지
column|expression	명명된 열 또는 표현식을 선택
alias	선택된 열에 다른 머리글을 지정
FROM table	열을 포함하는 테이블을 지정

- 테이블의 이름을 기록한 테이블 명세

```
SELECT * FROM tab;
```

- 전체 테이블 명세

```
SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno FROM emp;
SELECT * FROM emp;
```

- 특정 열 선택

```
SELECT ename, sal FROM emp;
```

- 주석

```
SELECT * /*주석*/ FROM emp;  -- 주석
```

- dual

DUAL은 함수 및 계산의 결과를 볼 때 사용할 수 있는 공용 (public) 테이블

사용자 데이터가 있는 테이블에서 유래하지 않은 상수 값,
의사열(pseudo-column), 표현식 등의 값을 단 한번만 돌려 거나 현재 날짜,
시각을 알고자 할 때 이용된다. 즉 일시적인 산술, 날짜 연산등에 주로 이용

```
SELECT ASCII(0) FROM dual;
48

SELECT ASCII('A') FROM dual;
65

SELECT SYSDATE FROM dual;

SELECT 7 + 10 FROM dual;
```

- 산술식 : 산술 연산자(+,-,*,/)를 사용하여 숫자 및 날짜 데이터로 표현식을 작성

```
SELECT ename, sal, sal + 300 FROM emp;

연산자 우선순위
SELECT ename, sal, 12*(sal + 300) FROM emp;
```

- NULL값 정의 : NULL은 사용할 수 없거나, 할당되지 않았거나, 알 수 없거나, 적용할 수 없는 값(NULL은 0이나 공백과는 다름)

```
SELECT empno, ename, job, comm FROM emp;
```

- 산술식의 null 값 : null 값을 포함하는 산술식은 null로 계산

```
SELECT ename, 12*sal*comm FROM emp;
```

- 열 alias 정의

열 이름을 바꿈
열 이름 바로 뒤에 나옴. 열 이름과 alias 사이에 선택 사항인 as 키워드가 올 수도 있음
알리아스에 큰따옴표를 사용하는 경우
- 대소문자 구별을 원할 때
- 공백 포함시

- _,# 등 특수문자 사용시(_는 “” 없이 중간에는 올 수 있으나 맨 앞에 오면 오류 발생) part_one 인정, _one 불인정 -> “_one” 인정)
- 숫자로 시작할 경우

```
SELECT sal*12 ASal FROM emp;
SELECT sal*12 as ASal FROM emp;
SELECT sal*12 “Annual Salary” FROM emp;
SELECT ename as name, comm commission FROM emp;
SELECT ename "Name", sal*12 "Annual Salary" FROM emp;
```

- 연결 연산자

```
열이나 문자열을 다른 열에 연결
두 개의 세로선(||)으로 나타냄
결과 열로 문자식을 생성

SELECT ename || ' has $' || sal FROM emp;
SELECT ename || job as "Employees" FROM emp;
```

- 연결 연산자와 null 값 : 문자열에 null 값을 결합할 경우 결과는 문자열

```
ename || null 의 결과는 ename

SELECT ename || comm FROM emp;

리터럴 문자열

리터럴은 select 문에 포함된 문자, 숫자 또는 날짜
날짜 및 문자 리터럴 값은 작은 따옴표로 묶어야 함
각 문자열은 각 행(row)이 반환될 때 한 번씩 출력

SELECT ename || ' is a ' || job as "Employee Details" FROM emp;
SELECT ename || ' : 1 Month salary = ' || sal Monthly FROM emp;
```

- distinct : 중복행 삭제

```
SELECT DISTINCT deptno FROM emp;
```

- 테이블 구조 표시 : describe 명령을 사용하여 테이블의 구조를 표시

```
DESC[RIBE] tablename
```

```
DESCRIBE emp;
DESC emp;
```

2) WHERE : 선택을 사용하여 행 제한

```
select {*|[distinct] column|expression [alias],...}
from table
[where logical expression(s)];

where 절은 열의 값, 리터럴, 산술식 또는 함수를 비교할 수 있으며 다음 세 가지 요소로 구성
- 열 이름
- 비교 조건
- 열 이름, 상수 또는 값 리스
- 알리아스는 사용할 수 없음
```

```
SELECT * FROM emp WHERE deptno=10;
```

- 문자열 및 날짜

```
문자열 및 날짜 값은 작은 따옴표로 묶음
문자 값은 대소문자를 구분하고 날짜 값은 형식을 구분함

- 문자열
SELECT ename, job, deptno from emp WHERE ename = 'SMITH';

- 날짜
SELECT *  from emp WHERE hiredate>'81-12-03';

[주의] where절에서는 알리아스를 사용할 수 없음
SELECT ename,sal,sal*12 ansal FROM emp WHERE sal*12 > 15000;
```

- 비교 연산자의 사용

```
select * from employees where hire_date > '08-01-13';
select * from employees where hire_date <>(or ^=, !=) '08-01-13';
select * from emp where sal > 2000 and sal < 5000;
```

- between ... and … : 두 값 사이(지정한 값 포함)

```
select * from emp where sal (not) between 1000 and 1500; (이상 ~ 이하)

select ename from emp where ename between 'KING' and 'SMITH';
```

- in : 값 목록 중의 값과 일치

```
select * from emp where sal (not) in (1300,2450,3000);

select ename, mgr, deptno from emp
where ename in ('ALLEN','FORD');
```

- like 연산자를 사용하여 패턴 일치

```
%는 0개 이상의 문자를 나타냄
_은 한 문자를 나타냄

select * from emp where ename (not) like '%S%'; (S가 처음, 중간, 끝에 오는 이름)

select * from employees
where ename (not) like 'S%'; (S로 시작하는 이름)

select ename, hiredate from emp
where hiredate like '%22';(22로 끝나는 입사일)

select * from emp where ename like 'FOR_';(FOR 다음에 꼭 한 글자)

select * from emp where ename like '_M%'; (한 글자 다음의 M, M 다음에 없거나 여러개)

검색하고자 하는 문자열에 _가 포함되어있을 경우 _ 앞에 \를 붙이고 escape '\'을
명시해서 검색
select * from emp where ename like 'SC\_%' escape '\';

검색하고자 하는 문자열에 %가 포함되어있을 경우
select * from emp where ename like '%\%' escape'\';
```

- NULL 조건 사용

```
select * from emp where comm is null; (comm =null은 불가)

select * from emp where comm is not null;(comm !=null은 불가)
```

- 논리 연산자(and,or,not)를 사용하여 조건 정의

```
AND 연산자의 사용 : 구성 요소 조건이 모두 TRUE 이면 TRUE를반환

select empno, ename, job, sal from emp where sal >= 2000
and job like '%MAN%';
```

OR 연산자의 사용 : 조건 중 하나가 TRUE면 TRUE를 반환

```
select empno, ename, job, sal from emp
where sal >= 2000
or job like '%MAN%';
or job_id like '%MAN%';
```

NOT 연산자의 사용

```
select ename, job from emp
where job not in ('CLERK','SALESMAN');
```

- 우선 순위 규칙

순위	연산자 혹은 조건
1	+(양수),-(음수)
2	*, /
3	+(더하기),-(나누기)
4	=, !=, <,>,<=,>= (비교)
5	is [not] null, like, [not] between and, [not] in, exists (비교)
6	not (부정)
7	and (논리곱)
8	or (논리합)

3) ORDER BY :정렬

구문

```
  SELECT    expr
  FROM      table
  [WHERE      condition(s)]
  [ORDER BY {column, expr, numeric_position} [ASC|DESC]];
```

구문 설명:

ORDER BY 검색된 행이 표시되는 순서를 지정합니다.
ASC 오름차순으로 행을 정렬합니다(기본 순서).
DESC 내림차순으로 행을 정렬합니다.

```
select * from emp where deptno=10 order by sal asc;  <-default
                                                    order by sal desc;
                                                    order by sal, ename;
                                                    order by sal asc, ename desc;
```

- 내림차순 정렬

```
SELECT   ename, job, deptno, hiredate
FROM     emp
ORDER BY hiredate DESC ;
```

- 열 alias를 기준으로 정렬

```
SELECT empno, ename, sal*12 annsal FROM   emp
ORDER BY annsal ;
```

- 열의 숫자 위치를 사용하여 정렬

```
SELECT   ename, job, deptno, hiredate FROM     emp
ORDER BY 3;

테이블에 생성된 순서대로 열 숫자 위치 부여
SELECT * FROM emp ORDER BY 2;
```

- 여러 열을 기준으로 정렬
1차 정렬의 기준열에 중복값이 있으면 2차 정렬이 가능

```
SELECT ename, deptno, sal FROM   emp
ORDER BY deptno, sal DESC;
```

주: NULLS FIRST 또는 NULLS LAST 키워드를 사용하여 반환된 행 중 null 값을 포함하는 행이 정렬 순서상 맨 처음에 나타나거나 마지막에 나타나도록 지정

```
select * from emp order by comm nulls  first;
select * from emp order by comm nulls last;

select * from emp order by comm nulls last, ename desc;

select * from emp order by comm desc nulls  first; ← nulls first 뒤에 desc를 명시하면 오류
select * from emp order by comm desc nulls last;
```

4) 함수

- 단일행 함수

문자함수

대소문자 조작 함수 - LOWER, UPPER, INITCAP

```
select LOWER ('HELLO') from dual; -> 소문자 hello 로 변경

select UPPER ('hello') from dual; -> 대문자 HELLO 로 변경

select INITCAP ('hello wORLD') from dual; -> Hello World 로 변경

select INITCAP (ename) from emp; -> 문장의 첫글자를 대문자로 변경, 그 다음은 소문자로 변환
```

- 문자 조작함수(Character Functions)

CONCAT (문자열1, 문자열2) : 문자열1과 문자열2를 연결하여 하나의 문자열로 반환

```
select CONCAT ('Hello', 'World') from dual; -> 문자열과 문자열2 연결
```

SUBSTR(대상문자열, 인덱스) : 대상문자열에서 지정한 인덱스부터 문자열을 추출
[주의]인덱스 1부터 시작

```
select SUBSTR ('Hello World',3) from dual; -> llo World  문자열 추출

select SUBSTR ('Hello World',3,3) from dual; -> llo 문자열 세번째부터 3개만 추출

select SUBSTR ('Hello World',-3) from dual; -> rld 뒤에서 3번째부터 끝까지 추출

select SUBSTR ('Hello World',-3,2) from dual; -> rl 뒤에서 3번째부터 2개만 추출
```

SUBSTRB(대상문자열, 인덱스) : 대상문자열에서 지정한 인덱스(바이트로 환산)부터 문자열을 추출

```
select SUBSTRB ('홍길동',7) from dual; -> 동  문자열 추출
```

LENGTH (대상문자열) : 문자열의 개수

```
select LENGTH ('Hello World') from dual; -> 11
```

LENGTHB(대상문자열) : 문자열의 바이트 수

```
select LENGTH ('홍길동') from dual; -> 9
```

INSTR(대상문자열, 검색문자) : 검색문자의 위치값 검색

```
select INSTR ('Hello World','e') from dual; -> 2

검색문자가 없을 경우 0
select INSTR ('Hello World','E') from dual; -> 0

select INSTR ('Hello World','o') from dual; -> 5

select INSTR ('Hello World','o',6) from dual; -> 8 (대상문자열, 검색문자,
검색인덱스:해당위치부터 검색)

select INSTR ('Hello World','o',1,2) from dual; -> 8 (대상문자열, 검색문자,
검색인덱스,반복횟수:*대상문자열 전체를 여러번 검색한다는 의미가 아니라 1번 검색으로
o를 찾으면 지정한 횟수만큼 그 뒤의 문자를 검색한다는 의미)
```

LPAD(대상문자열,총길이,문자) :지정한 길이에 문자열을 출력하는데 공백은 왼쪽에 지정한 문자로 채움
RPAD (대상문자열,총길이,문자) : 지정한 길이에 문자열을 출력하는데 공백은 오른쪽에 지정한 문자로 채움

```
select LPAD ('Hello',10,'*') from dual; -> *****Hello (대상문자열,총길이,문자)
select RPAD ('Hello',10,'*') from dual; -> Hello***** (대상문자열,총길이,문자)
```

TRIM :문자열에서 공백이나 특정 문자를 제거한 다음 값을 반환

```
select TRIM (both 'h' from 'habchh') from dual; -> abc
(방향(leading(왼쪽),trailing(오른쪽),both<-default) 제거문자 from 대상문자)
select TRIM (leading 'h' from 'habchh') from dual; -> abchh

select TRIM (both 'h' from 'hahchh') from dual; -> ahc
```

LTRIM(대상문자열,제거할 문자) :문자열의 왼쪽에서 공백이나 특정 문자를 제거한 다음 값을 반환

```
select LTRIM ('habchh', 'h') from dual; -> abchh
```

RTRIM :문자열의 오른쪽에서 공백이나 특정 문자를 제거한 다음 값을 반환

```
select RTRIM ('habchh', 'h') from dual; -> habc
```

REPLACE (대상문자열,old,new) : 대상문자열에서 old문자를 new문자로 대체

```
select REPLACE ('010.1234.5678','.','-') from dual; -> 010-1234-5678
```

함수 중첩

```
SELECT ename, LOWER(SUBSTR(ename,1,3)) FROM emp;
```

숫자 함수(Number Functions)

- CEIL(실수) : 올림 처리한 정수값을 반환

```
select ceil(1.4) from dual;
```

- FLOOR(실수) : 버림 처리한 정수값을 반환

```
select floor(1.7) from dual;
```

- ROUND (대상숫자, 지정자릿수) : 반올림

```
select ROUND (45.926,2) from dual; -> 45.93

select ROUND (45.926) from dual; -> 46

SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923, -  1)
FROM   DUAL;

select empno, ename, sal, round(sal*1.15,0) "New Salary", round(sal*1.15,0) - sal "Increase"
from emp;
```

- TRUNC(대상숫자,지정자릿수) : 절삭

```
select TRUNC (45.926,2) from dual; -> 45.92

select TRUNC (45.926) from dual; -> 45

SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923, -  1)
FROM   DUAL;
```

- MOD (대상숫자, 나눌 숫자) :나머지값

```
select MOD (17,2) from dual; -> 1    17%2는 오류

짝수 해와 홀수 해에 입사한 사원의 정보 구하기
SELECT CASE MOD(EXTRACT(YEAR FROM hiredate),2)
      WHEN 0 THEN
        '짝수년도'
      ELSE
        '홀수년도'
      END AS YEAR
     ,COUNT(empno) AS employee_number
FROM emp
GROUP BY MOD(EXTRACT(YEAR FROM hiredate),2);
```

- 날짜함수(Date Functions)

오라클 데이터베이스는 내부 숫자 형식(세기, 년, 월, 일, 시, 분, 초)으로 날짜를 저장. 기본 날짜 표시 형식은 DD-MON-RR

- 연도의마지막 두 자릿수만지정하면 21세기 날짜를 20세기에 저장할 수 있습니다.
- 같은 방식으로 20세 날짜를 21세기에 저장할수 있습니다

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-08';
```

RR 날짜 형식

현재 연도	지정된 날짜	RR형식	YY형식
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

2001	27-OCT-95	1995	2095

		지정된 두 자리 연도가 다음과 같을 경우 :	
		0 – 49	50 – 99
현재 연도의 두 자리가 다음과 같을 경우 :	0 – 49	반환 날짜는 현재 세기의 날짜입니다 .	반환 날짜는 이전 세기의 날짜입니다 .
	50 – 99	반환 날짜는 이후 세기의 날짜입니다 .	반환 날짜는 현재 세기의 날짜입니다 .

현재 연도	지정된 날짜	해석 (RR)	해석(YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2048	27-OCT-52	**1952**	2052
2051	27-OCT-47	**2147**	2047

- SYSDATE(ORACLE 서버의 현재 날짜와 시간을 반환)

```
select sysdate from dual;
```

날짜에 산술 연산자 사용

```
SELECT ename, (SYSDATE -  hiredate)/7 AS WEEKS FROM   emp
WHERE deptno = 10;
```

- MONTHS_BETWEEN(날짜1,날짜2) :두 날짜 간의 월 수

```
select months_between('2024-02-23','2023-01-23') from dual;
13 (날짜,날짜:월 간격)
```

```
select ename, trunc(months_between(sysdate,hiredate),0) months_worked from emp order
by months_worked;
```

- ADD_MONTHS
특정 날짜의 월에 정수를 더한 다음 해당 날짜를 반환하는 함수

```
select add_months('2024-01-01',8) from dual;
-> 2024-09-01 (날짜,숫자)
```

- NEXT_DAY : 지정된 요일의 다음 날짜

```
SELECT NEXT_DAY ('2024-03-13','월요일') FROM dual;
->24/03/18 (날짜,문자: 돌아오는 다음요일의 날짜)

1(일요일) ~ 7(토요일)
select NEXT_DAY ('2024-03-13',6) from dual;
-> 24/03/15
```

- LAST_DAY : 월의 마지막 날

```
SELECT LAST_DAY ('2024-03-16') FROM dual;
```

-ROUND (두번째 파라미터로 지정된 값을 기준으로 해서 첫번째 파라미터 날짜를 반올림)

ROUND()를 사용할 때 반올림 되는 형식
연도는 7월1일 이상일 경우에 올림, 그 이전 날짜의 경우는 잘려나감.
월은 16일 이상일 경우 올리고 그 이전 날짜의 경우는 잘려나감
분기는 3개월 단위이므로, 두번째 월의 16일 이상일 경우 올림, 그 이전 날짜의 경우는
잘려나감.
예) 1~3 > 2월16일, 4~6 > 5월 16일, 7~9 > 8월 16일, 10~12 > 11월16일

```
select round(sysdate) from dual;
Or
select round(sysdate,’DD’) from dual;  <- 시간이 12을 넘어서면 1을 증가시킴

select round(sysdate,'month') from dual; <- 16일 이상일 경우 월을 1 증가시킴

select round(sysdate,'year') from dual; <- 7월1일 이상일 경우 연도를 1 증가시킴

select round(sysdate,'q') from dual; “분기” <- 분기 두번째 달의 16일 이상일 경우 1 증가시킴
Select round(to_date(‘17/06/03’),’year’) from dual;
Select round(sysdate,’day’) from dual; <- day : 한 주가 시작되는 날짜
```

- TRUNC()

```
select TRUNC (sysdate) from dual;
select TRUNC (sysdate,'year') from dual;
select TRUNC (sysdate,'month') from dual;
```

- EXTRACT() 함수
날짜 정보에서 특정한 연도, 월, 일,시간, 분, 초 등을 추출

```
SELECT EXTRACT(YEAR FROM SYSDATE),
        EXTRACT(MONTH FROM SYSDATE)
        ,EXTRACT(DAY FROM SYSDATE)
FROM dual;
```

변환 함수(Conversion Functions)

암시적 데이터 유형 변환

할당문의 경우 Oracle server는 다음을 자동으로 변환

소스	대상
VARCHAR2 또는 CHAR	NUMBER
VARCHAR2 또는 CHAR	DATE
NUMBER	VARCHAR2 또는 CHAR
DATE	VARCHAR2

표현식 계산을 위해 Oracle Server는 다음을 자동으로 변환

소스	대상
VARCHAR2 또는 CHAR	NUMBER
VARCHAR2 또는 CHAR	DATE

명시적 데이터 유형 변환
TO_CHAR : 숫자 -> 문자, 날짜 -> 문자
TO_NUMBER : 문자 -> 숫자
TO_DATE : 문자 -> 날짜

날짜 포맷팅

TO_DATE()와 TO_CHAR(날짜) 함수에서 사용할 수 있는 포맷팅 옵션

AD, A.D. - AD 표시
AM, PM, A.M.,P.M. - 오전/오후 표시
BC, B.C. -> BC 표시
RM - 월의 로마식 표기
CC, SCC - 세기 표시(1999는 20세기로, 2001은 21세기로 표현됨)
Y,YY,YYY,YYYY - 연도의 숫자값(각각 1,2,3,4 자리로 표현)
YEAR - 연도를 문자로 표현(예, “Two thousand one”)
RR - 네 자리 연도 숫자 중 뒤의 두 자리를 사용하여 앞의 두 자리 숫자를 반환. 뒤의 두 자리가 50보다 작을 때는 현재 세기가 그대로 사용되지만, 50이상이면 1을 줄여 사용한다는 것에 주의 (예) RR(‘99’) = 1999, RR(‘00’) = 2000
MON - 월의 세 자리 문자식 표현(JAN,FEB 등)
MONTH - 월의 문자식 표현(JANUARY, FEBRUARY 등)
MM - 월의 숫자식 표현(1~12)
WW - 주간의 숫자식 표현(1~53)
W - 월 내의 주간을 숫자식으로 표현(1~5)
D - 주간 내의 일을 숫자식으로 표현(1~7)
DD - 월 내의 일을 숫자식으로 표현(1~31)
DDD - 연도 내의 일을 숫자식으로 표현(1 ~ 365)
Day - 주간 내의 일을 문자식으로 표현
HH, HH12 - 시간(1~12)
HH24 - 24시간식 표현(0~23)
MI - 분(0~59)
SS - 초(0~59)
SSSS - 자정 이후 하루 내의 초 단위(0~86399)

Q - 분기(1,2,3,4) 1은 1월에서 3월까지

숫자 포맷팅

9 - 숫자를 주어진 자리수대로 반환(예, TO_CHAR(111,999) = 111)
0 - 0 이 표시되도록 강제 적용
9,999 - 정해진 위치에 콤마를 넣는다.(예, TO_CHAR(1234,’9,999’) = 1,234)
999.00 - 정해진 위치에 소수점을 넣는다.(예, TO_CHAR(123,’999.99’) = 123.00)
$9999 - 숫자 앞에 달러 기호를 넣는다.
FM99 - 리턴된 문자값의 앞에 있는 공백을 지운다.
RN, rn - 주어진 숫자를 로마 숫자로 표기(각각 소문자와 대문자)
X - 주어진 숫자를 16진수로 표기(예, TO_CHAR(20,’XX’) = 14)

- TO_CHAR

select to_char (날짜,'포맷문자') from dual;

```
select to_char (sysdate,'YYYY-MM-DD') from dual;

select to_char(sysdate, 'DD-MON-RR HH:MI:SS') from dual;

select to_char(sysdate, 'fmDD-MON-RRfm HH:MI:SS') from dual; (fm ~ fm 선행zero와 공백제거)

select to_char(sysdate, 'WW') from dual;

select to_char(sysdate, 'WWsp') from dual; - 기수화

select to_char(sysdate, 'WWspth') from dual; - 서수화
```

select to_char (수자,'포맷문자') from dual

```
실제 자리수와 일치
SELECT TO_CHAR(1234,9999) FROM dual;
SELECT TO_CHAR(1234,'9999') FROM dual;
SELECT TO_CHAR(1234,'0000') FROM dual;
##으로 출력 오류 발생
```

```
SELECT TO_CHAR(1234,0000) FROM dual;

자리수가 모자람-> ####
SELECT TO_CHAR(1234,999) FROM dual;
SELECT TO_CHAR(1234,'999') FROM dual;
SELECT TO_CHAR(1234,'000') FROM dual;

실제 자리수 보다 많은 자리수 지정
SELECT TO_CHAR(1234,99999) FROM dual;    1234
SELECT TO_CHAR(1234,'99999') FROM dual;   1234
SELECT TO_CHAR(1234,'00000') FROM dual; 01234

소수점 자리
SELECT TO_CHAR(1234,9999.99) FROM dual;  1234.00
SELECT TO_CHAR(1234,'9999.99') FROM dual;  1234.00
SELECT TO_CHAR(1234,'0000.00') FROM dual;  1234.00

반올림해서 소수점 둘째자리까지 표시
SELECT TO_CHAR(25.897,'99.99') FROM dual; 25.90
인상된 급여를 소수점 첫째자리까지 표시
SELECT TO_CHAR(sal*1.15,'9,999.9') FROM emp;

통화 표시
SELECT TO_CHAR(1234,'$0000') FROM dual; $1234
지역통화 표시
SELECT TO_CHAR(1234,'L0000') FROM dual; \1234
---------------------------------------------------------------------
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

- TO_DATE

```
select to_date('12-11-05','YYYY-MM-DD') from dual;
select to_date('12-11-05') from dual; 포맷형식 생략 가능

select to_date('12-11-05','DD-MM-RR') from dual;
```

RR 날짜 형식으로 TO_CHAR 및 TO_DATE 함수 사용

```
SELECT last_name, TO_CHAR(hire_date, 'DD  -  Mon  -  YYYY')
FROM employees
WHERE hire_date < TO_DATE('01 -  Jan  -  90','DD -  Mon  -  RR');
```

- TO_NUMBER

```
select to_number('100','999' or '000') from dual;

select to_number('100') from dual; 포맷형식 생략 가능

select to_number('-100.001') from dual;
```

- 일반 함수

- NVL (value1,value2) : value1이 null이면 value2를 쓴다. value1과 value2의 자료형이 일치

```
select ename, sal, nvl(comm,0), (nvl(comm,0)*12*sal)+(sal*12) from emp;

select ename, nvl(to_char(comm), 'No Commission') "COMM." from emp;
```

- NVL2 (value1,value2,value3) : value1이 null인지 평가. null이면 value3, null이 아니면 value2
자료형이 일치하지 않아도 됨

```
select nvl2(comm,'commission','no commission') from emp;

[주의]
select nvl2(comm,sal,comm) from emp; O
select nvl2(comm,'commission','no commission') from emp;  O
select nvl2(comm,comm,'no commission') from emp; X
select nvl2(comm,to_char(comm),'no commission') from emp; O
select nvl2(comm,'commission',0) from emp; O
```

- NULLIF (value1,value2) : 두개의 값이 일치하면 NULL. 두개의 값이 일치하지 않으면 value1

```
select NULLIF(length(ename),length(job)) "NULLIF" from emp;
```

- COALESCE (value1,value2,value3....) : null값이 아닌 값을 사용 (자료형 일치)

```
select comm, sal, coalesce(comm,sal,0) from emp;
select comm,mgr, sal, coalesce(comm,mgr,sal) from emp;
```

- case (Ansi) 컬럼 when 비교값 then 결과값
when then
when then
(else 결과값)

end

```
select ename, sal, job,
        case job when 'SALESMAN' then sal*0.1
                 when 'MANAGER' then sal*0.2
                 when 'ANALYST' then sal*0.3
                 else sal*0.4
        end "Bonus"
from emp;

select ename, sal, job,
        case when sal>=4000 and sal<=5000 then 'A'
             when sal>=3000 and sal<4000 then 'B'
             when sal>=2000 and sal<3000 then 'C'
             when sal>=1000 and sal<2000 then 'D'
             else 'F'
        end "Grade"
from emp order by sal DESC;
```

- DECODE (ORACLE 전용) : **=** 비교만 가능 decode(컬럼, 비교값,반환값,
  비교값,반환값,
  비교값,반환값,
  반환값)

```
select ename, sal, job,
    decode(job, 'SALESMAN',sal*0.1,
                'MANAGER',sal*0.2,
                'ANALYST',sal*0.3,
                sal*0.4)
            "Bonus"
from emp;

select ename, sal, job,
        decode(trunc(sal/1000), 5,'A',
                        4,'A',
                          3,'B',
                          2,'C',
                          1,'D',
                          'F')
        "Grade"
from emp  order by “Grade”, sal DESC;
```

- 특정 컬럼의 특정값을 먼저 오게 정렬하는 방법

order by (case 컬럼명 when 비교값 then 1(순서)
when 비교값 then 2
else 3
end)
order by decode (컬럼명,비교값,1,
비교값,2,
3)
사번으로 정렬하는데 7698인 경우는 가장 먼저 정렬

```
SELECT empno,ename,sal
FROM emp
ORDER BY (CASE empno WHEN 7698 THEN 1
               END), empno;

SELECT empno,ename,sal
FROM emp
ORDER BY DECODE(empno,7698,1), empno;
```

5) 그룹 함수

그룹 함수는 행 집합 연산을 수행하여 그룹별로 하나의 결과를 산출

- AVG() : NULL을 제외한 모든 값들의 평균을 반환. NULL값은 평균 계산에서 무시됨

```
select avg(sal) from emp;

NVL 함수는 강제로 그룹 함수에 null 값이 포함되도록 합니다 .
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

- COUNT() : NULL을 제외한 값을 가진 모든 레코드의 수를 반환. COUNT(*) 형식을 사용하면 NULL도 계산에 포함

```
select count(*) from emp;

select count(comm) from emp;
```

- MAX() : 레코드 내에 있는 여러 값 중 가장 큰 값을 반환

```
select max(sal) from emp;
SELECT MAX(ename) FROM emp;
SELECT MAX(hiredate) FROM emp;
```

- MIN() : 레코드 내에 있는 여러 값 중 가장 작은 값을 반환

```
select min(sal) from emp;
SELECT MIN(ename) FROM emp;
```

- SUM() : 레코드들이 포함하고 있는 모든 값을 더하여 반환

```
select sum(sal) from emp;
```

```
select max(sal), min(sal), round(avg(sal)), sum(sal) from emp;

select max(sal), min(sal), round(avg(sal)), sum(sal) from emp where deptno=10;

select count(*) from emp where deptno=20;

select count(ename) from emp where deptno=30;

select count(nvl(comm,0)) from emp;<- null값까지 카운트 하고 싶을 경우
```

6) GROUP BY & HAVING

select 절에 집합함수 적용시 개별 컬럼을 지정할 수 없다.
개별 컬럼을 지정할 경우에는 반드시 group by 절에 지정된 컬럼만 가능

```
select deptno, max(sal), min(sal), round(avg(sal)), sum(sal) from emp group by deptno;

select max(sal), deptno, job from emp group by deptno, job order by deptno;
```

다중 열에서 GROUP BY 절 사용

```
SELECT   deptno, job, SUM(sal)
FROM     emp
WHERE  deptno > 10
GROUP BY deptno, job
ORDER BY deptno; ;
```

그룹 함수를 사용한 잘못된 query

```
[오류]
SELECT deptno, COUNT(ename)
```

```
FROM   emp;

[수정]
SELECT deptno, COUNT(ename) FROM   emp
GROUP BY deptno;
```

그룹 처리 결과를 제한하고자 할 때 having절 이용
where절에는 집합함수를 사용할 수 없고 having절을 이용

```
오류발생
select deptno, avg(sal) from emp
where avg(sal)>=2000(group에 대한 제한을 할 수 없다)
group by deptno;

정상 구문
select deptno, avg(sal) from emp group by deptno having avg(sal) >=2000;
```

having절의 이용(having절에 알리아스 사용 x)

```
SELECT   deptno, MAX(sal)
FROM     emp
GROUP BY deptno
HAVING   MAX(sal)>3000;
```

그룹 함수 중첩

```
SELECT MAX(AVG(sal)) FROM emp GROUP BY deptno;
2916.66666666666666666666666666666666666667
```

그룹함수 이용하기

```
2007년 입사한 사원들의 목록으로부터 분기별 입사자의 수를 구함
SELECT TO_CHAR(hire_date,'Q') AS Quarter
     ,COUNT(employee_id) AS employee_number
FROM employees
WHERE EXTRACT(YEAR FROM hire_date) = 2007
GROUP BY TO_CHAR(hire_date,'Q');

분기별로 입사한 사원의 수
EXTRACT() 함수를 이용하여 입사 일자로부터 월을 구함. 3으로 나눈 값을 올림 처리하여 각 월의 분기를 계산
```

```
SELECT CEIL(EXTRACT(MONTH FROM hiredate)/3) AS Quarter
            ,COUNT(empno) AS count_member
FROM emp
GROUP BY CEIL(EXTRACT(MONTH FROM hiredate)/3)
ORDER BY quarter;

Or

SELECT TO_CHAR(hiredate,'Q') AS quarter,COUNT(empno) AS count_member
FROM emp
GROUP BY TO_CHAR(hiredate,'Q')
ORDER BY quarter;

2007년 3월 입사한 사원을 대상으로일주일 간격으로 입사자의 수를 구함
SELECT TRUNC(EXTRACT(DAY FROM hire_date)/7)+1 AS number_week
     ,COUNT(employee_id)
FROM employees
WHERE TO_CHAR(hire_date,'YYYY-MM') = '2007-03'
GROUP BY TRUNC(EXTRACT(DAY FROM hire_date)/7)+1
ORDER BY number_week ASC;
```

- 분석함수

RANK()
순위를 표현할 때 사용하는 함수

```
RANK(조건값) WITHIN GROUP (ORDER BY 조건값 컬럼명 [ASC | DESC]) : 특정 데이터의 순위 확인하기
(주의) RANK 뒤에 나오는 데이터와 ORDER BY 뒤에 나오는 데이터는 같은 컬럼이어야 함.
SELECT RANK('SMITH') WITHIN GROUP (ORDER BY ename) "RANK" FROM emp;
SELECT ename FROM emp ORDER BY ename; ← SMITH가 10번째 조회됨

-----------------------------------------------
RANK() OVER ([query_partition_clause] order_by_clause)
OVER절 다음 순위를 만들려면 정렬(ORDER BY)은 필수이며 그룹을 나누어 (PARTITION BY) 순위를 만드는 경우는 선택 사항

전체순위보기 : RANK() 뒤가 WITHIN GROUP 가 아니고 OVER로 바뀜
사원들의 empno, ename, sal, 급여 순위를 출력
SELECT empno, ename, sal, RANK() OVER (ORDER BY sal) AS RANK_ASC, RANK()
OVER (ORDER BY sal DESC) AS RANK_DESC FROM emp;

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

emp 테이블에서 10번 부서에 속한 직원들의 사번과 이름, 급여, 해당 부서 내의 급여 순위를
```

출력
SELECT empno, ename, sal, RANK() OVER (ORDER BY sal DESC) "RANK" FROM emp WHERE deptno = 10;

Emp 테이블을 조회하여 사번, 이름, 급여, 부서번호, 부서별 급여 순위를 출력
SELECT empno, ename, sal, deptno, RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) "RANK" FROM emp;

Emp테이블을 조회하여 empno, ename, sal, deptno,job, 같은 부서 내 job별로 급여 순위를 출력
SELECT empno, ename, sal, deptno,job, RANK() OVER (PARTITION BY deptno, job ORDER BY sal DESC) "RANK" FROM emp;

7) JOIN

둘 이상의 테이블을 연결하여 데이터를 검색하는 방법.
보통 둘 이상의 행들의 공통된 값 Primary Key 및 Foreign Key 값을 사용하여 조인
두 개의 테이블을 select문장 안에서 조인하려면 적어도 하나의 컬럼이 그 두 테이블 사이에서 공유되어야 함.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	80/12/17	800	(null)	20
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
7566	JONES	MANAGER	7839	81/04/02	2975	(null)	20
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
7698	BLAKE	MANAGER	7839	81/05/01	2850	(null)	30
7782	CLARK	MANAGER	7839	81/06/09	2450	(null)	10
7788	SCOTT	ANALYST	7566	87/04/19	3000	(null)	20
7839	KING	PRESIDENT	(null)	81/11/17	5000	(null)	10
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
7876	ADAMS	CLERK	7788	87/05/23	1100	(null)	20
7900	JAMES	CLERK	7698	81/12/03	950	(null)	30
7902	FORD	ANALYST	7566	81/12/03	3000	(null)	20
7934	MILLER	CLERK	7782	82/01/23	1300	(null)	10

emp 테이블

dept 테이블

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

(1) Cartesian Product(카티션 곱)
검색하고자 했던 데이터뿐 아니라 조인에 사용된 테이블들의 모든 데이터가 반환되는 현상

```
select * from emp, dept;
```

(2) Cartesian product는 다음과 같은 경우 발생

- 조인 조건을 정의하지 않았을 경우
- 조인 조건이 잘못된 경우
- 첫 번째 테이블의 모든 행들이 두 번째 테이블의 모든 행과 조인이 되는 경우

(3) join 방법

<Oracle 전용>
- 동등 조인(equi join)
조건절 Equality Condition(=)에 의하여 조인이 이루어 짐

```
select emp.ename, dept.dname
from emp, dept
where emp.deptno=dept.deptno;
-----------------------------------

테이블에 알리아스 부여하기

select e.ename, d.dname
from emp e, dept d
where e.deptno=d.deptno;

컬렴명을을 호출할 때 테이블명 또는 테이블 알리아스를 생략
select ename, e.deptno, dname
from emp e, dept d
where e.deptno=d.deptno;
—----------------------------

추가적인 조건 명시하기

select e.ename, d.dname
from emp e, dept d
where e.deptno=d.deptno
and e.ename='ALLEN';
-----------------------------------
select e.ename, e.sal, d.dname
from emp e, dept d
```

```
where e.deptno=d.deptno
and e.sal between 3000 and 4000;
```

- 비동등 조인(non equi join)
테이블의 어떤 column도 join할 테이블의 column에 일치하지 않을 때 사용하고 조인 조건은 동등(=)이외의 연산자를 갖음.(between and, is null, is not null, in, not in)

```
사원이름,급여,급여등급 구하기(emp,salgrade 테이블 이용)
select e.ename,e.sal,s.grade
from emp e, salgrade s
where e.sal between s.losal and s.hisal;
```

- Self Join

```
사원 이름과 해당 사원의 관리자 이름 구하기(관리자가 없는 사원 제외)
select e.ename 사원이름, m.ename 관리자이름
from emp e, emp m
where e.mgr=m.empno;
```

- 외부 조인(Outer Join)
equi join 문장들의 한 가지 제약점은 그것들이 조인을 생성하려 하는 두 개의 테이블의 두 개 컬럼에서 공통된 값이 없다면 테이블로부터 데이터를 반환하지 않는다는 것.
정상적으로 조인 조건을 만족하지 못 하는 행들을 보기위해 outer join을 사용

```
누락된 행의 반대 테이블에 (+)기호 표시
select distinct(e.deptno), d.deptno
from emp e, dept d
where e.deptno(+) = d.deptno;

사원이름과 해당 사원의 관리자 이름구하기(관리자가 없는 사원도 표시)
SELECT a.ename 사원이름, m.ename 관리자이름
FROM emp a, emp m
WHERE a.mgr = m.empno(+);
```

<표준 SQL>

- 내부 조인(inner join)

```
inner join 이라고 해도 되고 join만 명시 가능
select emp.ename, dept.deptno
from emp inner join dept
```

```
on emp.deptno = dept.deptno;
-----------------------------------
join 사용시 on절을 정의하고 부가적인 조건이 있으면 where절 사용
select e.ename, d.dname
from emp e join dept d
on e.deptno=d.deptno
where e.ename='ALLEN';
-------------------------------------------

만약 조인 조건에 사용된 컬럼의 이름이 같다면 다음과 같이 using절을 사용하여 조인
조건을 정의할 수 있음
select e.ename, d.dname
from emp e join dept d
using(deptno)
where e.ename='ALLEN';

[주의]using에 사용된 컬럼은 테이블명 또는 테이블 알리아스를 붙이지 않음
select e.ename, deptno ← d.deptno라고 하면 오류
from emp e join dept d
using(deptno)
where e.ename='ALLEN';

JOIN할 때 하나의 테이블에만 존재하는 컬럼은 테이블명 또는 테이블 알리아스를 붙이지
않아도 식별 가능
select ename, deptno,dname
from emp join dept
using(deptno);
```

- Self Join

```
사원 이름과 해당 사원의 관리자 이름 구하기(관리자가 없는 사원은 제외)
select e.ename name, m.ename manager_name
from emp e join emp m
on e.mgr = m.empno;
```

- 외부 조인(Outer Join)

```
누락된 행의 방향 표시
select distinct(e.deptno), d.deptno
from emp e right outer join dept d
on e.deptno = d.deptno;

사원 이름과 해당 사원의 관리자 이름 구하기(관리자 없는 사원도 표시)
SELECT e.ename name, m.ename manager_name
```

```
FROM emp e LEFT OUTER JOIN emp m
ON e.mgr = m.empno;
```

8) 집합연산자

- union (합집합 중복값 제거)
union은 두 테이블의 결합을 나타내며, 결합시키는 두 테이블의 중복되지 않은 값들을 반환

```
select deptno from emp
union
select deptno from dept;
```

- union all
union과 같으나 두 테이블의 중복되는 값까지 반환

```
select deptno from emp
union all
select deptno from dept;
```

- intersect
intersect는 두 행의 집합 중 공통된 행을 반환

```
select deptno from emp
intersect
select deptno from dept;
```

- minus
minus는 첫번째 select문에 의해 반환되는 행 중에서 두 번째 select문에 의해 반환되는 행에 존재하지 않는 행들을 보여줌

```
select deptno from dept
minus
select deptno from emp;
```

9) Subquery
다른 하나의 SQL 문장의 절에 nested된 select문장

- 단일행 서브쿼리 : 오직 한 개의 행(값)을 반환

```
Select empno, ename, job from emp
where job = (select job from emp where empno = 7369);

SELECT empno,ename,sal FROM emp
WHERE sal=(SELECT sal FROM emp WHERE empno=7654);

SELECT empno,ename,sal FROM emp
WHERE sal > (SELECT sal FROM emp WHERE empno=7698);
```

- 다중 행 서브쿼리
하나 이상의 행을 반환하는 서브쿼리

in 연산자의 사용

부서별로 가장 급여를 적게 받는사원과 동일한 급여를 받는 사원의 정보를 출력
```
select empno, ename,sal,deptno from emp
where sal in (select min(sal) from emp group by deptno);
```

any(some) 연산자의 사용

any 연산자는 서브쿼리의 결과값 중 어느 하나의 값이라도 만족이 되면 결과값을 반환

```
select sal from emp where job = 'SALESMAN'
select ename, sal from emp where sal> 1250 or sal > 1500 or sal>1600;
```
----> 서브쿼리로 표시
```
select ename, sal from emp
where sal> any(select sal from emp where job = 'SALESMAN');

select ename, sal from emp
where sal> some(select sal from emp where job = 'SALESMAN');
```

all 연산자의 사용

서브 쿼리의 결과와 모든 값이 일치

```
select sal from emp where deptno=20
select empno,ename,sal,deptno from emp where sal  > 800 and sal > 2975 and sal > 3000;
```
----> 서브쿼리로 표시
```
select empno,ename,sal,deptno from emp
where sal > all(select sal from emp where deptno=20);
```

- 다중열 서브쿼리
서브 쿼리의 결과가 두 개 이상의 컬럼으로 반환되어 메인 쿼리에 전달하는 쿼리

```
SELECT empno, ename, sal, deptno
FROM emp
WHERE (deptno, sal) IN ( SELECT deptno, sal
                         FROM emp
                          WHERE deptno = 30 );

부서별로 가장 급여를 적게 받는 사원의 정보를 출력
(주의)다중열을 사용했기 때문에 부서별로 가장 급여를 적게 받는 사원 정보 출력 가능)
SELECT empno,ename,sal,deptno
FROM emp
WHERE (deptno,sal) IN (SELECT deptno,MIN(sal) FROM emp GROUP BY deptno);

SELECT empno, ename, sal, deptno FROM emp
WHERE (deptno,sal) = (SELECT deptno,sal FROM emp WHERE ename = 'SMITH');

부서별로 가장 급여를 적게 받는 사원과 동일 급여를 받는 사원의 정보를 출력
SELECT empno,ename,sal,deptno
FROM emp
WHERE sal IN (SELECT MIN(sal) FROM emp GROUP BY deptno);
```

- 인라인뷰
메인 쿼리의 FROM절을 서브 쿼리로 이용하는 방법

```
급여가 20번부서의 평균 급여보다 많은 급여를 받는 사원의 사원번호,이름,부서명 출력
SELECT e.empno,e.ename,d.dname FROM (SELECT *
                    FROM emp WHERE sal> (SELECT AVG(sal)
                                         FROM emp WHERE deptno=20)) e JOIN dept d
USING(deptno);

SELECT e.empno,e.ename,d.dname FROM emp e JOIN dept d
USING(deptno) WHERE sal> (SELECT AVG(sal)
                          FROM emp WHERE deptno=20);

부서별로 부서번호,부서명,급여총액을 출력하시오.
SELECT deptno,dname,total FROM dept JOIN
(SELECT deptno, SUM(sal) total FROM emp GROUP BY deptno)
USING(deptno);
```

- 스칼라 서브쿼리
* 스칼라값이란 벡터값에 대응되는 말로 단일 값을 의미함
결과값이 단일 행, 단일 열의 스칼라값으로 반환된다. 만약 결과값이 다중 행이거나 다중 열이라면 DBMS는 그 중 어떠한 행, 어떠한 열을 출력해야 하는지 알 수 없어 에러를 출력한다.

```
부서별 급여 합계를 구하고 부서번호,부서명,급여 합계를 출력하시오.
```

```
– 스칼라 서브쿼리 형태
SELECT deptno, (SELECT dname FROM dept WHERE deptno=e.deptno), sum(sal) FROM
emp e GROUP BY deptno;
```

2-4 INSERT 문

테이블에 행을 삽입
형식 : INSERT INTO 테이블명 (컬럼명....) VALUES (값....)

```
전체 데이터 삽입(전체 컬럼 명시시)
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
values (8000,'DENNIS','SALESMAN',7698,'99/01/22',1700,200,30);

전체 데이터를 삽입할 때는 컬럼명 생략 가능
INSERT INTO emp
VALUES (8001,'SUNNY','SALESMAN',7698,'99/03/01',1000,300,30);

[null 삽입 방법]
값이 입력되지 않는 컬럼은 제외
insert into emp (empno,ename,job,mgr,hiredate,sal,deptno)
values (8003,'PETER','CLERK',7698,'12/11/06',1700,20);

값이 입력되지 않는 컬럼을 제외하지 않았을 경우
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
values (8004,'ANNIE','CLERK',7698,'12/11/06',1800,null,30);

날짜의 삽입
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
values (8005,'MICHAEL','CLERK',7698,TO_DATE('12/11/06','YY/MM/DD'),1800,null,30);

다른 테이블에서 행 복사
  ● VALUES는 사용하지 않음
  ● Insert 절의 열 수와 서브 쿼리의 열 수를 일치시킴

create table sales_reps(
id number(4) not null,
name varchar2(10) not null,
salary number(7,2) not null,
commission_pct number(7,2)
);

insert into sales_reps(id,name,salary,commission_pct)
select empno,ename,sal,comm from emp
```

```
where job like '%LES%';
```

2-5 UPDATE 문

행 단위로 데이터 갱신
형식 : UPDATE 테이블명 SET 컬럼명=데이터,... WHERE 조건

```
UPDATE emp SET mgr=7900 WHERE empno=8000;
UPDATE emp SET ename='MARIA',sal=2500,comm=500 WHERE empno=8000;

서브 쿼리로 두 열 갱신

update emp
set job = (select job
          from emp
          where empno = 7839),
    sal = (select sal
          from emp
          where empno = 7839)
where empno = 7369;
다른 테이블의 기반으로 행 갱신
update emp
set deptno = (select deptno
          from emp
          where empno = 7839),
where job =  (select job
                    from emp
                    where empno = 7839);
```

2-6 DELETE 문

행을 삭제
형식 : DELETE FROM 테이블명 WHERE 조건

```
delete from emp where empno=8000;

다른 테이블의 기반으로 행 갱신
delete from emp where deptno=(select deptno
                                        from emp
                                        where empno = 7839);
```

2-7 데이터베이스 트랜잭션

트랜잭션은 데이터 처리의 한 단위
오라클 서버에서 발생하는 SQL문들을 하나의 논리적인 작업단위로써 성공하거나 실패하는 일련의 SQL문을 트랜잭션이라고 할 수 있음
트랜잭션은 데이터를 일관되게 변경하는 DML문장으로 구성됨

1) 트랜잭션의 시작
실행 가능한 SQL문장이 제일 처음 실행될 때

2) 트랜잭션의 종료
COMMIT 이나 ROLLBACK
DDL이나 DCL문장의 실행(자동 COMMIT)
기계 장애 또는 시스템 충돌(crash)
deadlock 발생
사용자가 정상 종료

3) 자동 COMMIT은 다음의 경우 발생
DDL, DCL 문장이 완료 될 때
명시적인 COMMIT이나 ROLLBACK 없이 SQL*Plus를 정상 종료 했을 경우

4) 자동 ROLLBACK은 다음의 경우 발생
SQL*Plus를 비정상 종료 했을 경우
비정상적인 종료
system failure

5) COMMIT(변경사항 저장) & ROLLBACK(변경사항 취소)

- COMMIT과 ROLLBACK의 장점

- 데이터의 일관성을 제공
- 데이터를 영구적으로 변경하기 전에 데이터 변경을 확인하게 함
- 관련된 작업을 논리적으로 그룹화 함
- COMMIT, ROLLBACK 문장으로 트랜잭션의 논리를 제어

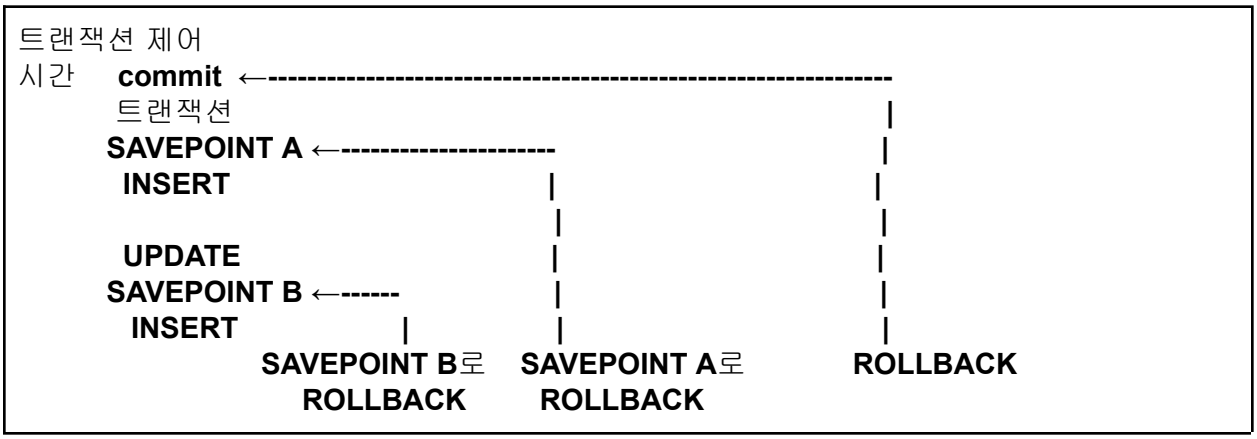
- COMMIT이나 ROLLBACK 이전의 데이터 상태
- 데이터 이전의 상태로 복구 가능
- 현재 사용자는 SELECT문장으로 DML작업의 결과를 확인
- 다른 사용자는 SELECT문장으로 현재 사용자가 사용한 DML문장의 결과를 확인할 수 없음
- 변경된 행은 **LOCK**이 설정되어서 다른 사용자가 변경할 수 없음

- COMMIT이후의 데이터 상태
- 데이터베이스에 데이터를 영구적으로 변경
- 데이터의 이전 상태는 완전히 상실

- 모든 사용자가 결과를 볼 수 있음
- 변경된 행의 LOCK이 해제되고 다른 사용자가 변경할 수 있음

6) 트랜잭션의 4가지 성질

원자성(Atomicity)	트랜잭션에 포함된 작업은 전부 수행되거나 아니면 전부 수행되지 않아야 all or noting 한다.
일관성(Consistency)	트랜잭션을 수행하기 전이나 수행한 후나 데이터베이스는 항상 일관된 상태를 유지해야 한다.
고립성(Isolation)	수행 중인 트랜잭션에 다른른트랜잭션이 끼어들어 변경 중인 데이터 값을 훼손하는 일이 없어야 한다.
지속성(Durabitity)	수행을 성공적으로 완료한 틀내잭션은 변경한 데이터를를영구히 저장해야 한다. 저장된 데이터베이스는 저장 직후 혹은 어느 때나 발생할 수 있는 정전, 장애 오류에 영향을 받지 않아야 한다.



명령문	설명
commit	보류 중인 모든 데이터 변경 내용을 영구히 저장하고 현재 트랜잭션을 종료
savepoint name	현재 트랜잭션 내에 저장점을 표시
rollback	rollback은 보류 중인 모든 데이터 변경 내용을 버리고 현재 트랜잭션을 종료
rollback to savepoint name	Rollback to savepoint는 현재 트랜잭션을 지정된 저장점으로 롤백하여 롤백하는 저장점 이후에 생성된 모든 변경 내용 및 저장점을 버림. to savepoint 절을 생략할 경우 rollback 문은 전체 트랜잭션을 롤백. 저장점은

	논리적인 것이므로 작성한 저장점을 나열할 수 없음. *savepoint는 ANSI표준 SQL이 아님. ex) **update …..** **savepoint update_done;** Savepoint created. **insert ….** **rollback to update_done;** Rollback complete.

3. 데이터베이스 객체

객체	설명
테이블	기본 저장 단위로 행과 열로 구성
뷰	논리적으로 하나 이상의 테이블에 있는 데이터의 부분 집합을 나타냄
시퀀스	숫자 값 생성기
인덱스	질의의 성능을 향상
동의어	객체에 다른 이름을 제공

3-1 테이블의 생성**,** 수정 및 삭제

1) 테이블이란?

(1) 테이블은 기본적인 데이터 저장 단위
(2) 레코드와 컬럼으로 구성

- 레코드(record, row) : 테이블의 데이터는 행에 저장
- 컬럼(column) : 테이블의 각 컬럼은 데이터를 구별할 수 있는 속성을 표현

2) 이름 지정 규칙
- 문자로 시작해야 함
- 1자부터 30자까지 가능
- A-Z, a-z, 0-9, _, $, #만 포함해야 함
- 동일한 사용자가 소유한 다른 객체의 이름과 중복되지 않아야 함.
- Oracle server의 예약어가 아니어야 함.

3) 오라클 데이터베이스의 테이블
- 사용자 테이블
  사용자가 생성 및 유지 관리하는 테이블의 collection

사용자 정보를 포함

- 데이터 딕셔너리
  Oracle Server가 생성 및 유지 관리하는 테이블의 collection
  데이터베이스 정보를 포함

접두어	설명
USER_	이 뷰는 사용자가 소유하는 객체에 관한 정보를 포함
ALL_	이 뷰는 사용자가 액세스할 수 있는 모든 테이블(객체 테이블 및 관계형 테이블)에 관한 정보를 포함
DBA_	이 뷰는 제한된 뷰로서 DBA 롤을 할당 받은 사용자만 액세스할 수 있음
V$	이들 뷰는 데이터베이스 서버 성능, 메모리 및 잠금에 대한 동적 성능 뷰

데이터 딕셔너리 질의

- 사용자가 소유한 테이블의 이름

```
SELECT table_name
FROM user_tables;
```

- 사용자가 소유한 개별 객체 유형

```
SELECT DISTINCT object_type
FROM user_objects;
```

- 사용자가 소유한 테이블, 뷰, 동의어 및 시퀀스

```
SELECT *
FROM user_catalog;
```

4) 테이블의 생성

- 테이블 이름 : 만들어질 테이블의 이름
- 열 이름 : 테이블 내에 만들어질 열의 이름. 열의 이름은 같은 테이블 내에서는 유일해야 하지만, 서로 다른 테이블 간에는 같은 이름을 사용할 수 있음
- 데이터 타입 : 각각의 열은 자신의 데이터 타입을 가진다. 열의 데이터 타입이 결정되면 어떤 데이터 타입이 지정되는지에 따라 데이터의 길이나 정확도와 스케일이 지정되어야 함
- **default <**표현식**>** : 각각의 열에는 insert 구문에 열의 값이 지정되지 않은 경우에 이용될 디폴트 값을 지정. <표현식> 부분에는 정적인 값이나 대부분의 SQL 함수를 지정 가능
- 제약조건 : 만들어질 각 열에 선택적으로 제약조건을 정의할 수 있다. 제약조건은 각 열의 값이 올바른 것이 되기 위해 지켜져야 할 규칙

```
예)
create table employee
(
  empno   number(6),
  name varchar2(30) not null,
  salary number(8,2),
  hire_date date default sysdate,
  constraint employee_pk_empno primary key (empno)
);

INSERT INTO employee (empno,name,salary) VALUES (100,'홍길동',1000.23);
COMMIT;
SELECT * FROM employee
```

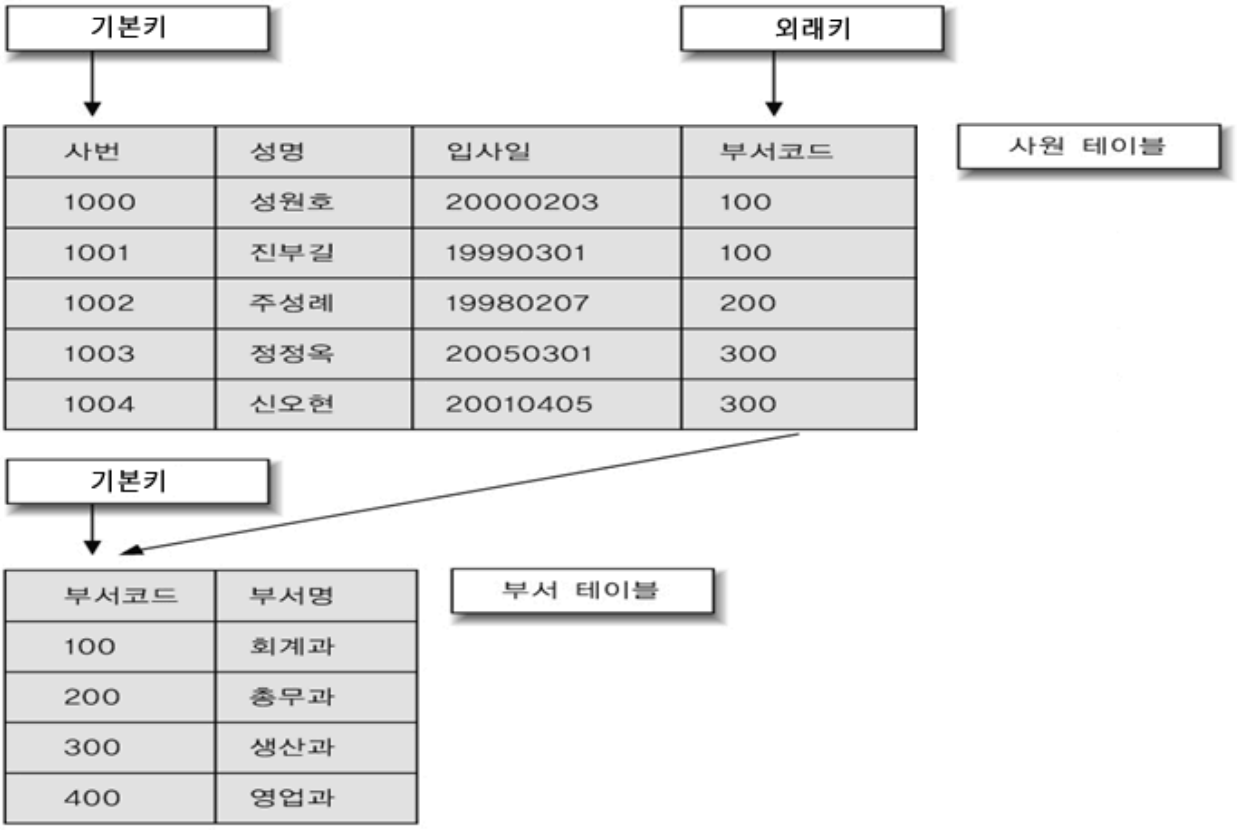
- 컬럼의 구조에 대한 상세정보를 조회

describe (or desc) 테이블명;

5) 제약 조건?

제약조건이란 테이블에 부적절한 자료가 입력되는 것을 방지하기 위해서 여러 가지 규칙을 적용해 놓은 것.

예) 기본키와 외래키



제약조건	설명
primary key(PK)	유일하게 테이블의 각 행을 식별(not null과 unique 조건을 만족)
foreign key(FK)	열과 참조된 열 사이의 외래키 관계를 적용하고 설정
unique key(UK)	테이블의 모든 행을 유일하게 하는 값(null 허용)
not null(NN)	열은 null값을 포함할 수 없음
check(ck)	해당 컬럼에 저장 가능한 데이터 값의 범위나 사용자 조건을 지정

- primary key & foreign key

```
create table suser(
id varchar2(20),
name varchar2(30),
constraint suser_pk_id primary key(id)
);

create table sboard(
```

```
num number,
id varchar2(20) not null,
content varchar2(4000) not null,
constraint sboard_pk_num primary key(num),
constraint sboard_suser_fk1 foreign key (id)  references suser (id)
);
```

Check

```
create table memo(
 no number constraint memo_chk CHECK(no>5)
);

insert into memo values (1);
--> 체크 제약조건(SCOTT.MEMO_CHK)이 위배되었습니다
insert into memo values (6);
--> 삽입
```

4) 오라클 데이터 타입

Data Type	설명
varchar2(n)	가변 길이 문자 데이터(1~4000byte)
char(n)	고정 길이 문자 데이터(1~2000byte)
number(p,s)	십진 자릿수 p고 소수점 이하 자릿수가 s인 숫자. 십진 자릿수의 범위는 1부터 38까지고 소수점 이하 자리수의 범위는 -84부터 127까지
date	날짜와 시간(초까지 표시) 기원전 4712년 1월 1일에서 기원후 9999년 12월 31일 사이의 날짜 및 시간 값
timestamp	날짜와 시간(밀리세컨드까지 표시)
long	가변 길이 문자 데이터(1~2gbyte)
clob	단일 바이트 가변 길이 문자 데이터(1~4gbyte)
raw(n)	n byte의 원시 이진 데이터(1~2000byte), 바이너리 데이터 저장
long raw	가변 길이 원시 이진 데이터(1~2gbyte)
blob	가변 길이 이진 데이터(1~4gbyte), 바이너리 데이터 저장

bfile	가변 길이 외부 파일에 저장된 이진 데이터(1~4gbyte)

5) 테이블의 관리

add 연산자 : 테이블에 새로운 컬럼을 추가

```
alter table employee add (addr varchar2(50));
```

제약 조건 추가

```
alter table employee add constraint employee_pk primary key (empno);
```

modify 연산자 : 테이블의 컬럼을 수정 하거나 not null 컬럼으로 변경 할 수 있음

```
alter table employee modify (salary number(10,2) not null);
```

drop 연산자 : 컬럼의 삭제

```
alter table employee drop column name;
```

컬럼명 변경

```
alter table employee rename column salary to sal;
```

테이블명 변경

```
rename employee to employee2;
```

6) 테이블의 삭제

```
drop table employee2;
```

7) on delete cascade
부모 테이블의 컬럼을 삭제하면 자식 테이블의 자식 데이터를 모두 삭제

```
create table s_member(
id varchar2(20) primary key,
name varchar2(30)
);
create table s_member_detail(
num number primary key,
content varchar2(4000) not null,
```

```
id varchar2(20) not null references s_member (id) on delete cascade
);
```

8) on delete set null
부모 테이블의 컬럼을 삭제하면 자식 테이블의 자식 데이터를 모두 null 처리

```
create table s_member(
id varchar2(20) primary key,
name varchar2(20)
);
create table s_member_detail2(
num number primary key,
text varchar2(4000) not null,
id varchar2(20) references s_member (id) on delete set null
);
```

3-2. 뷰**(View)**

논리적으로 하나 이상의 테이블에 있는 데이터의 부분 집합**.**
테이블을 JOIN하여 질의할 때 JOIN하는 테이블의 수가 늘어나면 질의문이 길고 복잡해질수 있다. 이럴 경우 View를 정의해서 사용하면 소스 코드에서 사용되는 질의문이 간결해지며 자연적으로 코드의 양도 줄어들어 유지보수가 편리한 장점이 있다.
뷰를 통해 테이블의 데이터를 보거나 변경할 수 있다. 뷰의 기반이 되는 테이블을 기본 테이블이라고 하며 뷰는 데이터 딕셔너리에 SELECT문으로 저장된다.

*뷰는 가상으로 만들어진 컬럼(virtual column)을 제외하면 수정이 가능하고 삭제도 가능하다 수정,삭제하면 원래 테이블에 반영, 삽입은 여러 제약 조건과 virtual column 사용으로 제약이 많음

- 뷰 사용 목적

```
- 데이터 액세스를 제한하기 위해
사원테이블(emp)의 컬럼을 보면 모든 사람에게 공개되어서는 안 될 급여와 같은 민감한 정보까지 포함하고 있는데 Oracle에서는 테이블에 저장되어 있는 데이터를 선별적으로 접근할 수 있는 권한까지는 부여하지 않는다. 따라서 각 부서 및 직책에 맞게 사원 테이블을 재생성해야 한다. 이럴 경우 View를 활용해서 공개를 원하지 않는 정보는 제외할 수 있다.

- 복잡한 질의를 쉽게 작성하기 위해
- 데이터 독립성을 제공하기 위해
- 동일한 데이터로부터 다양한 결과를 얻기 위해
```

- 사용중인 계정에 권한 이 없어서 View가 생성되지 않을 경우. 권한 부여하기

```
>sqlplus system/비밀번호
SQL>conn /as sysdba
SQL>grant create view to SCOTT; <- 사용중인 계정
```

1) View 생성

```
CREATE [OR REPLACE] VIEW 뷰이름 AS 쿼리;
```

```
CREATE OR REPLACE VIEW emp10_view
as SELECT empno id_number, ename name,
        sal*12 ann_salary
   FROM emp
   WHERE deptno=10;

select * from emp10_view;
----------------------------------------
CREATE OR REPLACE VIEW emp_info_view AS
SELECT e.empno, e.ename, d.deptno, d.loc, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;

select * from emp_info_view;
```

2) View의 구조 확인

```
SQL>DESC emp10_view;
------------------
SQL>DESC emp_info_view;
```

4) View를 통한 데이터 변경하기
일반적으로 View는 조회용으로 많이 사용되지만 아래와 같이 데이터를 변경할 수 있음

```
update emp10_view set name='SCOTT' where id_number=7839;
select * from emp10_view;
select * from emp; ← emp 테이블의 KING이 SCOTT로 변경됨

insert into emp10_view (id_number,name,ann_salary) values (8000,'JOHN',19000);
→ 가상 열은 사용할 수 없습니다 오류 발생
insert into emp10_view (id_number,name) values (8000,'JOHN'); ← 가상 열을 제외하면 삽입
가능
select * from emp10_view; ← 10번 부서만 보여지게 제한이 걸려서 삽입한 것이 안 보여짐
```

```
select * from emp; ← emp테이블에 1행이 추가됨

rollback;
```

5) With Read Only (읽기 전용 뷰를 생성하는 옵션)

```
CREATE OR REPLACE VIEW emp20_view
as SELECT empno id_number, ename name,
        sal*12 ann_salary
   FROM emp
   WHERE deptno=20
WITH READ ONLY;

update emp20_view set name='DAVID' where id_number=7902;
→ 읽기 전용 뷰에서는 DML 작업을 수행할 수 없습니다.
```

6) With check option (조건 컬럼값을 변경하지 못하게 하는 옵션)

```
CREATE OR REPLACE VIEW emp30_view
as SELECT empno, ename,deptno
   FROM emp
   WHERE deptno=30
with check option;

update emp30_view set deptno=10 WHERE empno=7499;
--> 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다 (WHERE deptno=30로 지정되어 있어
deptno 변경 불가능)
update emp30_view set ename='MARIA' WHERE empno=7499;
--> 수정 가능
delete from emp30_view WHERE deptno=30;
→ 삭제 가능

rollback;
```

7) View의 수정

```
CREATE OR REPLACE VIEW emp10_view
  (id_number,name,sal,department_id)
as select empno, ename,sal,deptno
   from emp
   where deptno=10;
참고 : CREATE VIEW 절에서 열 별칭을 지정하는 경우 서브 쿼리의 열과 동일한 순서로
나열해야 함.
```

8) View의 삭제

```
DROP VIEW emp10_view;
```

3-3. Sequence

유일한 값을 생성해주는 오라클 객체
시퀀스를 생성하면 기본키와 같이 순차적으로 증가하는 컬럼을 자동적으로 생성할 수 있음
보통 primary key 값을 생성하기 위해 사용

```
create sequence sequence_name
start with n
increment by n
maxvalue n | nomaxvalue
minvalue n | nominvalue
cycle | nocycle

start with : 시퀀스의 시작 값을 지정. n을 1로 지정하면 1부터 순차적으로 시퀀스번호가 증가
increment by : 시퀀스의 증가 값을 말함. n을 2로 하면 2씩 증가
                start with를 1로 하고 increment by를 2로 하면 1,3,5,7.... 이런식으로 시퀀스
                 번호 증가
maxvalue n | nomaxvalue : maxvalue는 시퀀스가 증가할 수 있는 최대값
                            nomaxvalue는 시퀀스의 값을 무한대로 지정
minvalue n | nominvalue : minvalue는 시퀀스의 최소값을 지정
                          기본값은 1이며, nominvalue를 지정할 경우 최소값은 무한대
```

```
create sequence test_seq
start with 1
increment by 1
maxvalue 100000;

시작 값이 1이고 1씩 증가하고, 최대값이 100000이 되는 시퀀스 생성

currval : 현재 값을 반환
nextval : 현재 시퀀스 값의 다음 값 반환

시퀀스의 수정

alter sequence sequence_name
increment by n
maxvalue n | nomaxvalue
minvalue n | nominvalue
cycle | nocycle

start with는 수정할 수 없음

시퀀스 삭제
drop sequence sequence_name;
```

3-4. 인덱스**(Index)**

인덱스는 데이터 검색을 빨리 하기 위해 사용한다**.** 하지만 인덱스를 생성했다고 해서 데이터 검색이 무조건 빨라지는 것은 아니다. 데이터의 양이 별로 없거나 데이터값이 몇 종류 안 되어 선택도가 높으면 인덱스가 없는 것이 더 빠를 수 있다.
***** 선택도(selectivity) 란 '1/서로 다른 값의 개수'를 말하는 것으로, 예를 들어 100개의 행을 가진 테이블에 값이 (남,여) 두 가지라면 선택도가 높다고 할 수 있다.

테이블의 컬럼에 대한 제약 조건을 설정할 때 Primary Key나 Unique로 설정하면 Oracle은 자동으로 이 컬럼에 대해 Unique Index를 설정한다. 데이터 값이 중복됨이 없이 유일한 값을 가지는 컬럼에 인덱스를 설정했을 때 이를 Unique Index라 한다.

1) 인덱스 만들기

- 자동 : 테이블 정의에 PRIMARY KEY 또는 UNIQUE 제약 조건을 정의하면 고유 인덱스가 자동으로 생성된다.
- 수동 : 사용자가 열에 고유하지 않은 인덱스를 생성하여 행에 대한 액세스 시간을 줄일 수 있다.

```
CREATE INDEX index_name
```

```
ON table_name (column_name[,column_name]...)
```

- 유일한 값을 가지는 컬럼에 인덱스 설정 : Unique Index

```
CREATE UNIQUE INDEX dname_idx
ON dept (dname);
```

- 유일한 값을 가지지 않는 컬럼에 인덱스 설정 : Non Unique Index

```
emp 테이블의 ename열에 대한 질의 액세스 속도 향상
create index emp_ename_idx
on emp(ename);
---------------------------
CREATE INDEX deptno_idx
ON emp (deptno);
```

2) 인덱스 생성이 필요한 경우

```
- 열에 광범위한 값이 포함된 경우
- 열에 널 값이 많이 포함된 경우
- WHERE 절 또는 조인 조건에서 하나 이상의 열이 함께 자주 사용되는 경우
```

3) 인덱스를 생성하지 않아야 할 경우

```
- 테이블이 작은 경우
- 열이 질의의 조건으로 자주 사용되지 않는 경우
- 테이블이 자주 갱신되는 경우
```

3-5. 동의어

동의어(객체의 다른 이름)를 생성하여 객체 액세스를 단순화한다.

```
- 다른 사용자가 소유한 테이블을 쉽게 참조
- 긴 객체 이름을 짧게 만든다.

CREATE [PUBLIC] SYNONYM synonym
FOR object;
```

1) 동의어 생성 및 제거

- 생성

```
CREATE SYNONYM emp20
FOR emp20_view;
-----------------------------------------
SELECT * FROM emp20
```

- 삭제

```
DROP SYNONYM emp20;
```

- 사용중인 계정에 권한이 없어서 SYNONYM가 생성되지 않을 경우. 권한 부여하기

```
>sqlplus system/비밀번호
SQL>conn /as sysdba
SQL>grant create synonym to SCOTT; <- 사용중인 계정
```

4. 권한

관리자 계정과 사용자 계정

데이터베이스의 계정은 관리자 계정과 일반 사용자 계정으로 나뉜다. 오라클에서 관리자 계정은 sys, system이고 일반 사용자 계정은 관리자 계정을 이용해서 생성하거나 기본 계정의 잠금 상태(LOCK)를 해제하여 사용하게 된다.

관리자 계정의 역할

데이터베이스의 생성과 관리를 담당하는 슈퍼유저**(Super User)**계정이며 데이터베이스 객체의 생성**,** 변경**,** 삭제 등의 작업이 가능하다. 오라클 데이터베이스 생성시 관리자 계정인 SYS와 SYSTEM 계정이 자동으로 생성된다. SYS 계정은 데이터베이스에서 발생하는 모든 문제를 처리할 수 있는 권한을 갖는다. SYSTEM 계정은 오라클 데이터베이스를 유지. 보수할 때 사용하는 계정이다. SYS와 달리 데이터베이스를 생성할 권한을 가지지 않는다.

관리자 계정은 데이터베이스 객체의 생성, 변경, 삭제 등을 할 수 있으며 또한 데이터베이스에 대한 모든 권한과 책임을 가지는 계정이다. 사용자를 생성하고 사용자에게 특정 시스템 권한을 부여하는 역할도 수행한다.

사용자 계정의 역할

사용자 계정은 데이터베이스에 접근하여 데이터를 조작**(**삽입**,** 삭제**,** 수정**,** 검색**)**하고 관리하는

일을 수행할 수 있는 계정이다. 일반 계정은 업무에 필요한 최소한의 권한만 가지는 것을 원칙으로 한다.

1) 시스템 권한
- 새 사용자 생성
- 사용자 제거
- 테이블 제거
- 테이블 백업

1-1) 사용자 생성
DBA는 CREATE USER문을 사용하여 사용자 생성

```
계정 생성
CREATE USER user
IDENTIFIED BY password;

CREATE USER scott
IDENTIFIED BY tiger;

접속권한부여
GRANT resource,connect TO user01;

[오라클 11버전]
Run SQL Command Line 선택

SQL>conn sys/1234(비밀번호) as sysdba
SQL>create user user002 identified by 1234(비밀번호) ← 계정 생성
SQL>GRANT resource,connect TO user002; ← 접속 권한 부여

[오라클 21버전]
SQL Plus 선택
사용자명 입력:system
비밀번호 입력:1234
SQL>conn /as sysdba
SQL>create user c##user002 identified by 1234(비밀번호) ← 계정 생성
SQL>GRANT resource,connect TO c##user002; ← 접속 권한 부여
```

1-2) 사용자 시스템 권한
DBA는 생성된 사용자에게 특정 시스템 권한을 부여할 수 있다.

```
GRANT privilege [,privilege... ]
TO user [,user|role,PUBLIC...];
GRANT create session, create table,
```

```
        Create sequence, create view
TO scott;
```

일반적인 사용자 권한
DBA는 생성된 사용자에게 권한을 할당할 수 있다.

시스템 권한	승인된 작업
CREATE SESSION	데이터베이스에 연결한다.
CREATE TABLE	사용자의 스키마에 테이블을 생성한다.
CREATE SEQUENCE	사용자의 스키마에 시퀀스를 생성한다.
CREATE VIEW	사용자의 스키마에 뷰를 생성한다.
CREATE PROCEDURE	사용자의 스키마에 내장 프로시저, 함수 또는 패키지를 작성한다.

2) 롤이란?
롤은 사용자에게 부여할 수 있는 관련 권한을 하나로 묶어 명명한 그룹으로서 롤을 사용하면 권한 취소 및 유지 관리를 쉽게 수행할 수 있다.
사용자는 여러 롤을 액세스할 수 있으며 동일한 롤이 여러 사용자에게 할당될 수도 있다.

2-1) 롤 생성 및 권한 부여

- 롤 생성

```
CREATE ROLE manager;
```

- 롤에 권한을 부여

```
GRANT create table, create view
TO manager;
```

- 사용자에게 롤을 부여

```
GRANT manager TO DEHAAN, KOCHHAR;
```

3) 암호 변경
DBA는 사용자 계정을 생성하고 암호를 초기화

```
ALTER USER scott
IDENTIFIED BY lion;
```

4) 사용자 삭제

```
DROP USER scott;
```

Scott 유저가 객체를 소유하고 있을 경우에는 CASCADE 옵션을 추가해서 삭제해야 함
CASCADE를 사용하게 되면 사용자 이름과 관련된 모든 데이터베이스 스키마가 데이터 사전으로부터 삭제되며 모든 스키마 객체들 또한 물리적으로 삭제됨

```
DROP USER scott CASCADE;
```

5) 객체 권한
객체권한은 유저가 소유하고 있는 특정한 객체를 다른 사용자들이 액세스 하거나 조작할 수 있게 하기 위해서 생성

객체 권한	테이블	뷰	시퀀스	프로시저
ALTER	O		O	
DELETE	O	O		
EXECUTE				O
INDEX	O	O		
INSERT	O	O		
REFERENCES	O	O		
SELECT	O	O	O	
UPDATE	O	O		

```
GRANT select
ON emp
TO sue, rich;
```

5) 객체 권한 취소
REVOKE문을 사용하여 다른 사용자에게 부여된 권한을 취소

```
REVOKE select, insert
```

```
ON dept
FROM scott;
```