오라클의 확장 SQL 문

그룹화

다중행 함수의 집계 관련 함수는 그룹화를 하지 않으면 일반 컬럼과 사용할 순 없다.

1. group by 문

group by 문을 사용하여 테이블의 컬럼을 그룹으로 나누고 각 그룹에 대한 요약 정보를 반환한다.
select 문에 그룹 함수를 포함하고 group by 문에 개별 컬럼을 지정하지 않는 경우 개별 결과도 선택할 수 없다.
where 문을 사용하면 컬럼을 그룹으로 나누기 전에 컬럼을 제외할 수 있다.
group by 문에는 별칭을 사용할 수 없다.
여러 개의 group by 문을 나열하여 그룹과 하위 그룹에 대한 요약 결과를 반환할 수 있다.
group by 문은 컬럼을 그룹화하지만, 결과에 대한 순서는 지정하지는 않으며 그룹화 순서를 지정하려면 order by 문을 사용한다.

형식:
select [distinct] *|columns [as 별칭] from tables
[where 조건식;]
group by column;
[order by column [asc/desc];]

설명:
select 명령어: 표시할 컬럼이나 명령어를 지정한다. 컬럼은 하나 이상의 컬럼으로 이루어진 리스트이다.
distinct 명령어: 중복을 방지한다.
*(애스터리스크) 기호: 모든 컬럼을 선택한다.
column 변수: 지정된 컬럼명을 설정한다.
[as 별칭]: 컬럼명의 별칭이나 가상 컬럼을 생성하며 as 명령어는 생략할 수 있다.
from tables: 컬럼을 포함하는 테이블들을 지정한다.
[where 조건식;]: 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한한다.

조건식은 조회할 조건에 대한 조건이나 표현식으로 ;(세미콜론)으로 종료한다.
group by column; : 테이블의 컬럼을 그룹으로 나누고 각 그룹에 대한 요약 정보를 반환하고 ;(세미콜론)으로 종료한다.
[order by column [asc/desc];] 문: 오름차순과 내림차순과 같은 정렬 방식을 설정한다. asc 명령어는 오름차순으로 정렬하고 desc 명령어는 내림차순으로 정렬하며 ;(세미콜론)으로 종료한다.

실습:
emp 테이블에서 사원번호로 그룹화하여 월급의 평균을 구한다.

```
select avg(sal) from emp
group by deptno;
```

2. having 문

having 문을 사용하여 표시할 그룹을 지정하면 집계 정보를 기초로 그룹을 추가로 제한할 수 있다.
having 문은 반환되는 그룹을 지정된 조건이 참인 그룹으로 제한한다.

having 문을 having 문을 사용할 때 다음 단계를 수행한다.
① 컬럼이 그룹화된다.
② 그룹 함수가 그룹에 적용된다.
③ having 문의 조건과 일치하는 그룹이 표시된다.

having 문이 group by 문 앞에 올 수 있지만 권장하지 않는다.
having 문이 그룹에 적용되기 전에 그룹이 형성되고 그룹 함수가 계산된다.

형식:
select [distinct] *|columns [as 별칭] from tables
[where 조건식;]
group by column
having 반환되는 그룹;
[order by column [asc/desc];]

설명:
select 명령어: 표시할 컬럼이나 명령어를 지정한다. 컬럼은 하나 이상의 컬럼으로 이루어진 리스트이다.
distinct 명령어: 중복을 방지한다.
*(애스터리스크) 기호: 모든 컬럼을 선택한다.
column 변수: 지정된 컬럼명을 설정한다.
[as 별칭]: 컬럼명의 별칭이나 가상 컬럼을 생성할 수 있다. as 명령어는 생략할 수 있다.
from tables: 컬럼을 포함하는 테이블들을 지정한다.
[where 조건식;] : 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한한다. 조건식은 조회할 조건에 대한 조건이나 표현식으로 ;(세미콜론)으로 종료한다.
group by column; : 테이블의 컬럼을 그룹으로 나누고 각 그룹에 대한 요약 정보를 반환하고 ;(세미콜론)으로 종료한다.
having 반환되는 그룹; : 그룹을 지정된 조건이 참인 그룹으로 제한한다.
[order by column [asc/desc];] 문: 오름차순과 내림차순과 같은 정렬 방식을 설정한다.

asc 명령어는 오름차순으로 정렬하고 desc 명령어는 내림차순으로 정렬하며 ;(세미콜론)으로 종료한다.

실습:

emp 테이블에서 급여가 500 이상인 사원번호와 급여를 구한다.

```
select max(sal) from emp
group by deptno
having max(sal) > 500;
```

시퀀스

시퀀스(sequence)는 값이 일정한 규칙에 따라서 연속적으로 자동 증가하는 오라클 객체이다.
시퀀스는 기본키(primary key)에서 사용하며 유일한 숫자를 자동으로 생성하는 자동 번호 발생기다.
시퀀스를 생성하면 기본키와 같이 순차적으로 증가하는 컬럼을 자동으로 생성할 수 있다.
시퀀스가 메모리에 캐시 되었을 때 시퀀스값의 액세스 효율이 향상한다.
시퀀스는 테이블과는 독립적으로 저장되고 생성되며 공유할 수 있다.

시퀀스는 다음 2 개의 제약조건을 사용하여 값을 관리한다.

- 시퀀스명.nextval : 시퀀스의 다음 값을 얻으며 nextval 은 next value 의 약자이다.
- 시퀀스명.currval : 시퀀스의 현재 값을 얻으며 currval 은 current value 의 약자이다.

시퀀스값을 삭제하거나 데이터베이스가 비정상적으로 종료하면 1, 2, 3, 5, 7 과 같이 시퀀스값에 간격이 생기고 번호가 중간에 점프하므로 일정하게 순차적 번호 관리를 할 수가 없다.
일정하게 순차적 번호를 관리해야 하는 프로그램에서는 사용할 수가 없다.
시퀀스는 주로 일정하게 순차적 번호 관리가 필요 없는 게시판 번호로 많이 사용된다.
시퀀스 형식의 마지막에는 ;(세미콜론)으로 종료한다.

형식:

```
create sequence 시퀀스명
[start with n]
[increment by n]
[maxvalue n]
[minvalue n]
[cycle]
[cache n];
```

설명:

create sequence 시퀀스명: 지정한 시퀀스명으로 시퀀스를 생성한다.
[start with n]: 시퀀스 번호의 시작값을 n 에 지정하고 생략하면 기본적으로 1 부터 시작한다.
[increment by n]: 시퀀스의 증가치를 n 에 지정하고 생략하면 기본적으로 1 씩 증가한다.
[maxvalue n]: 시퀀스가 가질 수 있는 최대값을 n 에 지정한다.
[minvalue n]: 시퀀스가 가질 수 있는 최소값을 n 에 지정한다.

[cycle]: 지정된 시퀀스 값이 최대값까지 증가가 완료되면 다시 초기값에서 시작한다.
[cache n]: 시퀀스가 관리할 캐시 개수를 n 에 지정하고 기본적으로 20 개를 관리한다.
myboard 테이블을 생성하고 myboard 테이블의 num 컬럼에 시퀀스를 부여한다.

실습 1:
myboard 테이블에 데이터를 입력한다.

```
create table myboard (
num number(4) primary key,
author varchar2(12),
title varchar2(30),
content varchar2(60)
);
create sequence myboard_seq;
```

실습 2:
myboard 테이블에 데이터를 입력한다.

```
insert into myboard(num, author, title, content)
values(myboard_seq.nextval, '전우치', '제목','내용이다.');
```

조인

1. 조인의 개요

조인(join)은 2 개 이상의 집합들을 연결하여 각 집합의 데이터들을 같이 다룰 수 있도록 하는 것이다.
집합에는 테이블, 뷰 형태가 올 수 있다.
집합을 연결하는 방법에 따라 기본적인 SQL 구성은 다음과 같이 분류할 수 있다.

A테이블

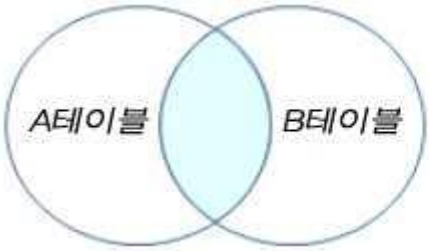
A키	A컬럼
1	가
2	나

B테이블

B키	B컬럼
1	100
3	1500
4	20

1) inner join

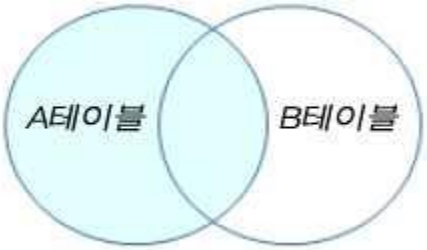
연결하려는 컬럼값이 일치하는 레코드만 가져온다.



A키	A컬럼	B키	B컬럼
1	가	1	100

2) left join

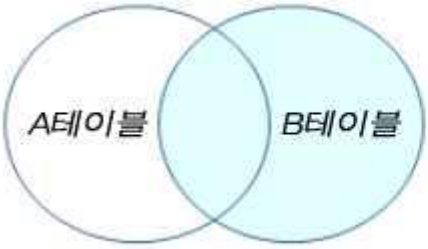
A 테이블이 조회 집합의 기준이 되고 A 테이블의 모든 데이터를 가져온다.
A 테이블의 A 키값과 일치하는 B 테이블의 B 키값이 있으며 B 테이블의 데이터도 같이 가져온다.
where 문의 조건으로 null 을 확인하고 B 테이블의 B 키값이 null 이면 A 테이블의 데이터만 가져온다.



A키	A컬럼	B키	B컬럼
1	가	1	100
2	나	null	null

3) right join

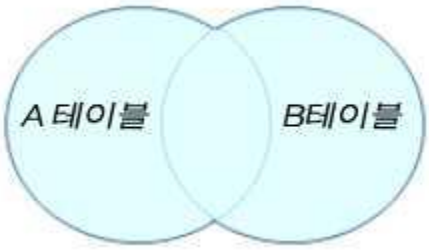
B 테이블이 조회 집합의 기준이 되고 B 테이블의 모든 데이터를 가져온다.
B 테이블의 B 키값과 일치하는 A 테이블의 A 키값이 있으며 A 테이블의 데이터도 같이 가져온다.
where 문의 조건으로 null 을 확인하고 A 테이블의 A 키값이 null 이면 B 테이블의 데이터만 가져온다.



A키	A컬럼	B키	B컬럼
1	가	1	100
null	null	3	1500
null	null	4	20

4) full outer join

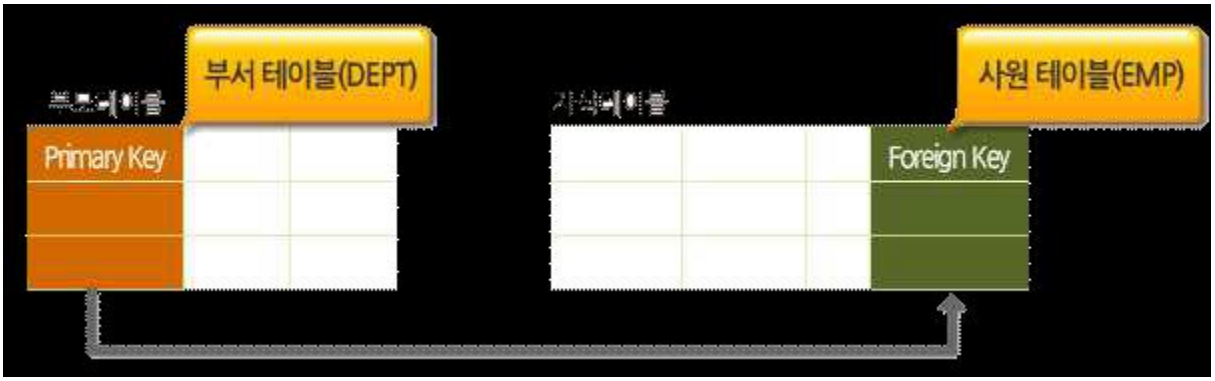
양쪽 테이블 모두가 조회 집합의 기준이 되며 양쪽 테이블의 데이터를 모두 가져온다.
연결 컬럼값이 일치하면 같은 레코드 상에 가져온다.
연결 컬럼값에 일치하는 값이 없으면 각각 다른 레코드 위치에 존재하면서 가져온다.
where 문의 조건으로 null 을 확인하고 A 테이블의 A 키값이 null 이거나 B 테이블의 B 키값이 null 이면 연결하려는 컬럼값이 일치하는 레코드를 제외하고 가져온다.



A키	A컬럼	B키	B컬럼
1	가	1	100
2	나	null	null
null	null	3	1500
null	null	4	20

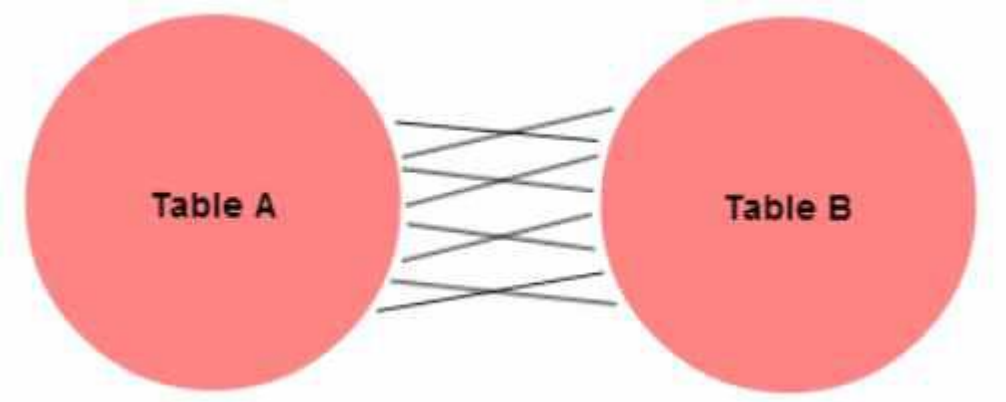
2. 오라클의 조인

조인은 일반적으로 관계형 데이터베이스에서는 하나 이상의 테이블 간에 관계를 설정하여 데이터를 조회한다.
일반적으로 기본키(Primary Key)와 외래키(Foreign Key)를 사용하여 조인하는 경우가 대부분이지만 때로는 논리적인 값들의 연관으로 조인할 수 있다.



관계형 데이터베이스에서는 테이블 간의 관계가 중요하기 때문에 하나 이상의 테이블이 빈번히 결합하여 사용되므로 한 개 이상의 테이블에서 데이터를 조회하기 위해서 조인을 사용한다.
오라클은 관계형 데이터베이스이므로 모든 정보가 하나의 테이블에 몰려 있는 것이 아니라 여러 테이블에 정규화되어 분산되어 있으며 각 테이블끼리는 서로 관계가 있도록 설계되어 있다.
오라클의 데이터는 여러 곳에 흩어져 있으므로 사용자가 원하는 데이터를 찾으려면 여러 테이블을 다 조회해야 한다.
조인은 여러 테이블에 흩어져 있는 정보 중에서 사용자가 필요한 정보만 가져와서 가상의 테이블처럼 만들어서 결과를 보여준다.
사원 테이블에서 이름을 가져오고 부서 테이블에서 부서명만을 가져와서 새로운 결과를 도출하는 것이 조인이다.

1) 크로스 조인



크로스 조인(Cross Join)은 특별한 컬럼명 없이 테이블을 ,(콤마)로 연결해서 사용한다.

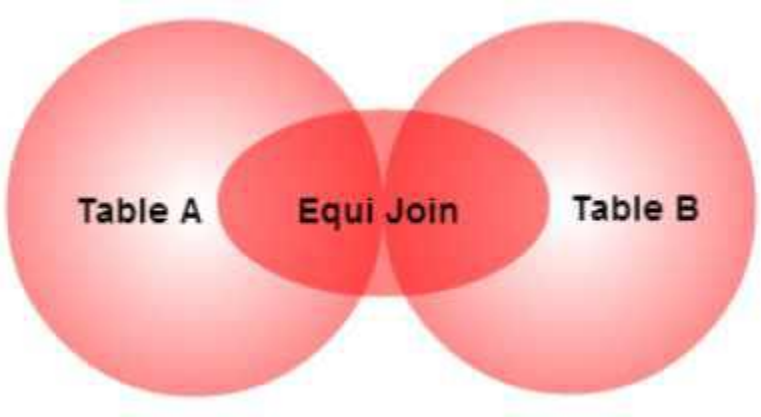
형식:
select * from table1, table2;

설명:
select 명령어: 표시할 컬럼이나 명령어를 지정한다.
*(애스터리스크) 기호: 모든 컬럼을 선택한다.
from table1, table2; : table1 과 table2 를 ,(콤마)로 연결해서 조인하고 ;(세미콜론)으로 종료한다.

실습:
emp 테이블과 dept 테이블을 크로스 조인을 한다.

select * from emp, dept;

2) 이퀴 조인



이퀴 조인(Equi Join)은 같은 컬럼을 기준으로 조인한다.
이퀴 조인은 가장 많이 사용하는 조인 방법으로서 조인 대상이 되는 두 테이블에서 공통으로 존재하는 컬럼값이 일치되는 컬럼을 연결하여 결과를 생성하는 조인 방법이다.
이퀴 조인은 조인 조건이 정확히 일치하는 경우 사용한다.
이퀴 조인은 조인 조건에서 =(이퀄) 연산자를 사용하여 컬럼값이 정확하게 일치하는 경우에 사용한다.
이퀴 조인은 다른 말로 단순조인 또는 내부조인이라고 하며 기본키와 외래키의 관계를 이용하여 조인한다.

(1) 테이블의 테이블명 사용
양쪽 테이블에 공통으로 존재하는 컬럼명은 모호함을 피하고자 컬럼명 앞에 반드시 테이블명을 기술한다.
서로 다른 테이블에 있는 같은 컬럼명에 접근하려고 하면 컬럼명이 중복되어 구별이 어려워 애매하다는 에러가 발생한다.

형식:

```
select table1.컬럼명, table2.컬럼명
from table1, table2
where table1.공통 컬럼명 = table2.공통 컬럼명;
```

설명:
select table1.컬럼명, table2.컬럼명: 테이블을 통해서 컬럼에 접근하여 조인한다.

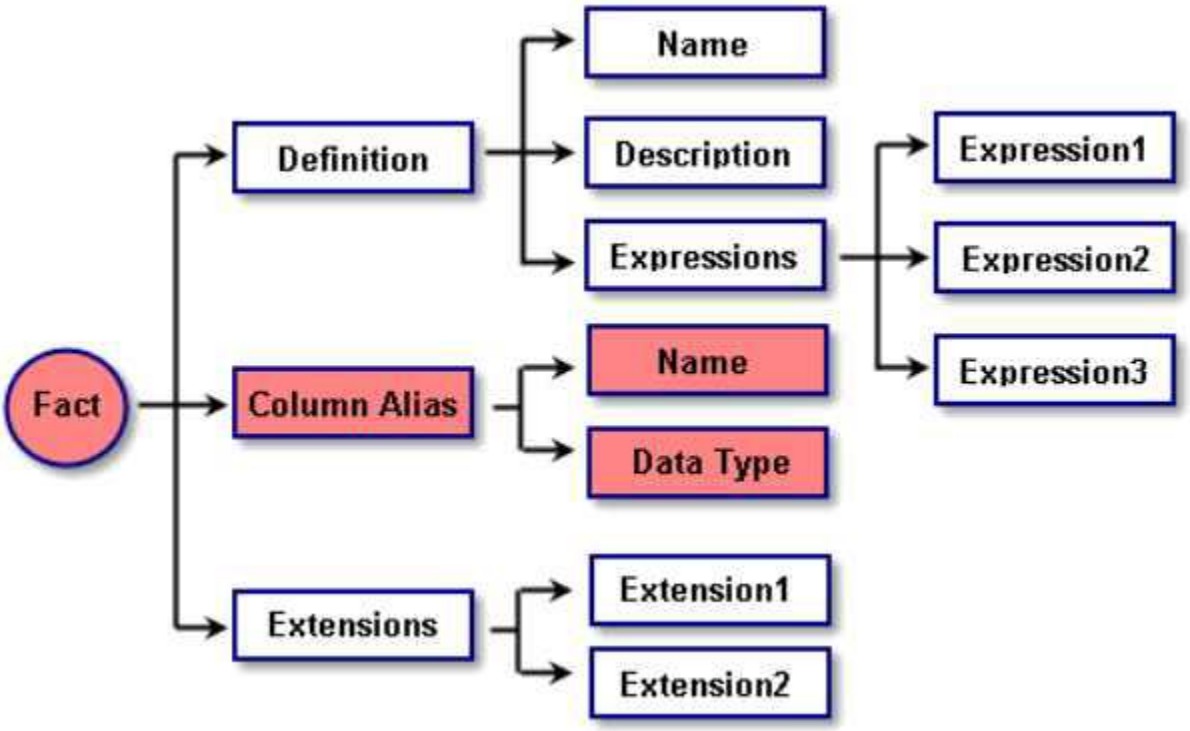
테이블을 조인할 때 명확성을 위하여 컬럼명 앞에 테이블명을 붙이는데 두 테이블에 같은 컬럼명을 사용하면 어느 테이블 소속인지 불분명하기에 오류가 발생한다.
같은 컬럼명 존재하는 테이블의 경우에는 반드시 컬럼명 앞에 테이블명을 붙인다.
from table1, table2: 조인할 테이블인 table1 과 table2 를 지정한다.
where table1.공통 컬럼명 = table2.공통 컬럼명; : =(이퀄) 연산자로 조인할 테이블의 컬럼명으로 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한하고 ;(세미콜론)으로 종료한다.

실습:
사원들의 사원번호를 조회한다.

```
select empno, ename, sal, dept.deptno
from emp, dept
where emp.deptno = dept.deptno;
```

3) 테이블의 별칭 사용

두 개 이상의 테이블을 조인할 때 컬럼명 앞에 테이블명을 기술하는 일이 잦아진다.
테이블에서 테이블명을 사용하여 여러 테이블에 있는 컬럼명을 한정하여 사용한다.
테이블명이 길면 테이블명을 매번 붙이기가 번거로워지므로 이런 번거로움을 제거하기 위해 테이블명 대신에 별칭(alias)을 사용한다.
테이블명에 간단한 별칭을 부여한 후에 컬럼명 앞에 테이블명 대신 간단한 별칭을 붙이면 문이 간단해 보인다.



테이블명의 별칭을 사용하여 이름은 같지만 서로 다른 테이블에 상주하는 컬럼을 구분한다.
테이블명의 별칭은 30 자까지 사용할 수 있지만 길이는 짧을수록 좋다.

형식:

select 별칭 1.컬럼명, 별칭 2.컬럼명

from table1 별칭 1, table2 별칭 2

where 별칭 1.공통 컬럼명 = 별칭 2.공통 컬럼명;

설명:

select 별칭 1.컬럼명, 별칭 2.컬럼명: 별칭을 통해서 컬럼에 접근하여 조인한다.

from table1 별칭 1, table2 별칭 2: 테이블의 별칭을 생성한다. 테이블명에 별칭을 붙이는 방법은 from 문 다음에 테이블명을 명시하고 공백 다음에 별칭을 설정하면 된다.

where 별칭 1.공통 컬럼명 = 별칭 2.공통 컬럼명; : =(이퀄) 연산자로 조인할 테이블의 컬럼명으로 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한하고 ;(세미콜론)으로 종료한다.

실습 1:

사원들의 사원번호와 소속 부서를 조회한다.

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

실습 2:

부서번호가 30 번인 사원번호, 이름, 급여, 부서명을 조회한다.

```
select e.empno, e.ename, e.sal, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and d.deptno = 30;
```

실습 3:

직급이 SALESMAN 사원의 사원번호, 이름, 급여, 부서명을 조회한다.

```
select e.empno, e.ename, e.sal, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and e.job ='SALESMAN';
```

실습 4:

커미션을 받는 사원의 이름, 직급, 부서 위치를 조회한다.

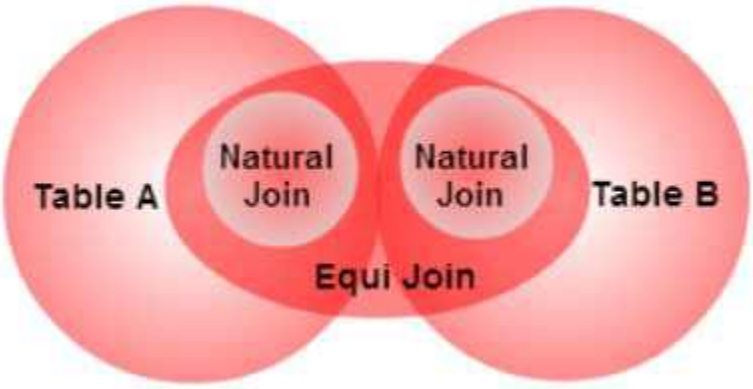
```
select e.empno, e.ename, e.sal, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and e.comm is not null;
```

실습 5:

이름에 'A' 문자가 들어있는 사원의 이름, 직급, 부서 위치를 조회한다.

```
select e.ename, e.job, d.loc
from emp e, dept d
where e.deptno = d.deptno and e.ename like '%A%';
```

4) 논 이퀴 조인



논 이퀴 조인(Non-Equi Join)은 같은 컬럼이 없이 다른 조건을 사용하여 조인한다.
논 이퀴 조인은 등급, 학점 등과 같이 조인 조건이 정확히 일치하지 않았을 때 사용한다.
논 이퀴 조인은 조인 조건에서 between A and B 연산자를 사용하여 컬럼값이 정확히 일치하지 않을 때 사용한다.
논 이퀴 조인은 기본키와 외래키의 관계를 이용하지 않고 조인한다.
기본키와 외래키는 조인을 위한 제약조건이 아니라 데이터 무결성을 위한 제약조건이다.
무결성이 필요한 대용량 데이터가 아닌 경우에는 기본키와 외래키를 사용하여 조인하는 것은 데이터 조작어인 SQL 문을 잘못 작성했을 때 수행에 문제가 발생할 수도 있다.
특별히 무결성을 확인할 필요가 없는 컬럼 간의 조인은 논 이퀴 조인으로 조인을 하여 실효성을 높이는 것이 더 효율적이다.

형식:
select 별칭 1.컬럼명, 별칭 2.컬럼명
from table1 별칭 1, table2 별칭 2
where 별칭 1.컬럼명 between 별칭 2.컬럼명 and 별칭 2.컬럼명;

설명:
select 별칭 1.컬럼명, 별칭 2.컬럼명: 별칭을 통해서 컬럼에 접근하여 조인한다.
from table1 별칭 1, table2 별칭 2: 테이블의 별칭을 생성한다. 테이블명에 별칭을 붙이는 방법은 from 문 다음에 테이블명을 명시하고 공백 다음에 별칭을 설정하면 된다.
where 별칭 1.컬럼명 between 별칭 2.컬럼명 and 별칭 2.컬럼명; : between A and B 연산자로 조인할 테이블의 컬럼명으로 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한하고 ;(세미콜론)으로 종료한다.

실습:

부서번호가 10 번인 부서에 대하여 사원번호, 이름, 업무, 급여, 급여의 등급을 조회한다.

```
select e.empno, e.ename, e.job, e.sal, s.grade
from salgrade s, emp e
where e.sal between s.losal and s.hisal and e.deptno = 10;
```

5) 셀프 조인

셀프 조인(Self Join)은 한 테이블 내에서 조인한다.
셀프 조인은 다른 테이블이 아닌 자기 자신과 조인을 한다.
셀프 조인은 별칭을 사용해서 마치 2 개의 서로 다른 테이블처럼 조인할 수 있다.
셀프 조인은 같은 테이블에 대해 두 개의 별칭을 작성하여 테이블을 구분함으로 from 절에 두 개의 테이블을 사용하는 것과 같다.
셀프 조인을 할 때는 컬럼에 대해서도 어떤 테이블에서 왔는지 별칭을 기술해야 한다.
셀프 조인은 테이블 하나를 두 개 또는 그 이상으로 조인할 수 있다.
셀프 조인은 하나의 테이블에서 컬럼을 조인하고자 할 때 사용한다.
셀프 조인은 하나의 테이블 내에서 조인해야만 원하는 자료를 얻을 때 사용한다.

형식:
select 별칭 1.컬럼명, 별칭 2.컬럼명
from table1 별칭 1, table1 별칭 2
where 별칭 1.컬럼명 = 별칭 2.컬럼명;

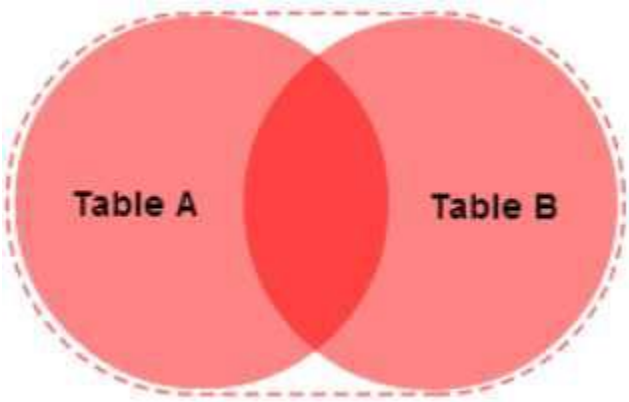
설명:
select 별칭 1.컬럼명, 별칭 2.컬럼명: 별칭을 통해서 컬럼에 접근하여 조인한다.
from table1 별칭 1, table1 별칭 2: 같은 테이블에서 서로 다른 별칭을 생성한다. 테이블명에 별칭을 붙이는 방법은 from 문 다음에 테이블명을 명시하고 공백 다음에 별칭을 설정하면 된다.
where 별칭 1.컬럼명 = 별칭 2.컬럼명; : =(이퀄) 연산자로 조인할 테이블의 컬럼명으로 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한하고 ;(세미콜론)으로 종료한다.

실습:

사원명과 사원을 담당하는 관리자명을 조회한다.

```
select employee.ename as 사원명, manager.ename as 관리자명
from emp employee, emp manager
where employee.mgr = manager.empno;
```

6) 아우터 조인



아우터 조인(Outer Join)은 조인 조건에 만족하지 않는 행도 나타낸다.
아우터 조인은 조인 조건이 정확히 일치하지 않을 때도 모든 행을 조회한다.
아우터 조인은 조인 조건에 만족하지 않아도 해당 행을 출력하고 싶을 때 사용할 수 있다.
아우터 조인은 조인할 데이터가 없는 쪽에 ( )(퍼렌씨시스) 안에 +(플러스) 연산자를 사용한다.
아우터 조인에 (+) 연산자를 조인 조건에 사용하면 조인 조건을 만족하지 않는 행들도 조회한다.
(+) 연산자는 한 개 이상의 null 행을 생성하고 정보가 충분한 테이블에 한 개 이상의 행들이 null 행에 조인된다.
(+) 연산자는 표현식의 한 편에만 올 수 있다.

형식:
select 별칭 1.컬럼명, 별칭 2.컬럼명
from table1 별칭 1, table1 별칭 2
where 별칭 1.컬럼명 = 별칭 2.컬럼명(+);

설명:
select 별칭 1.컬럼명, 별칭 2.컬럼명: 별칭을 통해서 컬럼에 접근하여 조인한다.
from table1 별칭 1, table1 별칭 2: 같은 테이블에서 서로 다른 별칭을 생성한다. 테이블명에 별칭을 붙이는 방법은 from 문 다음에 테이블명을 명시하고 공백 다음에 별칭을 설정하면 된다.

where 별칭 1.컬럼명 = 별칭 2.컬럼명(+); : =(이퀄) 연산자로 조인할 테이블의 컬럼명으로 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한하고 ;(세미콜론)으로 종료한다.
별칭 2.컬럼명(+) 문에서 (+) 연산자로 조인 조건에 만족하지 않아도 해당 행을 출력한다.

실습:
사원명과 사원을 담당하는 관리자명을 조회하고 조인 조건에 만족하지 않는 행도 모두 조회한다.

```
select employee.ename as 사원명, manager.ename as 관리자명
from emp employee, emp manager
where employee.mgr = manager.empno(+);
```

서브쿼리

1. 서브쿼리의 개요

형식:
select 컬럼명, 컬럼명 from table
where 조건식(select 컬럼명, 컬럼명 from table where 조건식);

서브쿼리(SUBQUERY)는 하나의 쿼리문의 결과를 다른 쿼리문에 전달하기 위해 두 개 이상의 쿼리문을 하나의 쿼리문으로 연결하여 처리하는 방법이다.
서브쿼리는 select 문에 내포된 select 문을 ( )(퍼렌씨시스) 안에 적용하여 사용할 수 있다.
사원인 SMITH 의 부서명을 조회하기 위해서는 사원인 SMITH 의 부서번호를 조회하고 나서 조회한 부서번호로 부서명을 조회할 수 있다.
서브쿼리를 적용하면 부서번호인 20 대신에 사원인 SMITH 의 부서번호를 조회하는 쿼리문을 기술하면 한 개의 쿼리문으로 부서명을 조회할 수가 있다.

```
select deptno from emp
where ename = 'SMITH';
```

⇨

```
select dname from dept
      where deptno = 20;
```

실행 결과

```
DEPTNO
----------
20
```

실행 결과

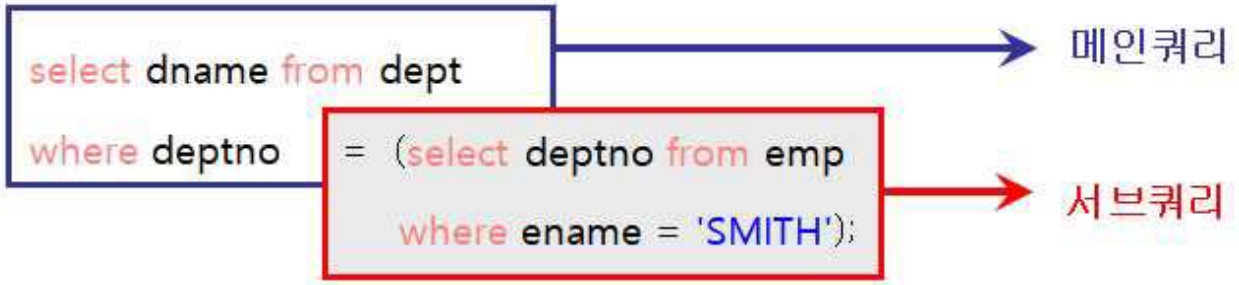
```
DNAME
-------------
RESEARCH
```

--> 서브 쿼리의 적용

```
select dname from dept
where deptno = (select deptno from emp where ename = 'SMITH');
```

실행 결과

```
DNAME
--------------
RESEARCH
```



서브쿼리는 메인쿼리가 실행되기 전에 한 번씩 실행되고 서브쿼리에서 실행된 결과가 메인 쿼리에 전달되어 최종적인 결과를 출력한다.
내포된 서브쿼리는 메인 쿼리 이전에 한 번만 수행되며 서브쿼리의 결과를 메인쿼리의 조건으로 제공하게 된다.
서브쿼리는 테이블 자체의 데이터에 의존하는 조건으로 테이블의 컬럼값을 조회할 필요가 있을 때 아주 유용하다.
서브쿼리는 다른 하나의 쿼리문에 포함된 select 문으로 명시되어 지지 않은 조건에 근거한 값들을 조회하는 select 문을 작성하는 데 유용하다.
서브쿼리의 결과는 메인 쿼리에 의해 조건 또는 데이터로 사용된다.

2. 서브쿼리의 유형

1) 단일행 서브쿼리

단일행 서브쿼리는 서브쿼리에서 단 하나의 행만을 검색하여 메인쿼리에 반환하는 질의문이다.
단일행 서브쿼리는 내부 select 문으로부터 하나의 행만을 조회한다.
단일행 서브쿼리는 서브쿼리의 결과로 하나의 행만이 출력되어야 한다.
메인쿼리의 where 문에서 서브쿼리의 결과와 비교할 때는 반드시 단일행 비교 연산자 중 하나만 사용해야 한다.

단일행 비교 연산자:

- =(이퀄) : 같다.
- !=(엑스클러메이션 포인트 이퀄) : 같지 않다.
- >(라이트 앵글브래킷) : 크다.
- >=(라이트 앵글브래킷 이퀄) : 크거나 같다.
- <(레프트 앵글브래킷) : 작다.
- <=(레프트 앵글브래킷 이퀄) : 작거나 같다.

(1) 단일행 연산자 적용
단일행 서브쿼리는 단일행 연산자를 사용하고 where 문에 기술된 컬럼의 개수와 데이터 타입은 select 문에 기술된 컬럼과 좌측으로부터 일대일 대응이 되며 데이터 타입이 일치해야 한다.

실습 1:
emp 테이블에서 SMITH 의 부서명을 호출한다.

```
select dname from dept
where deptno = (select deptno from emp where ename = 'SMITH');
```

실습 2:
emp 테이블에서 사원번호가 7698 인 사원의 급여보다 많은 사원의 사원번호, 사원명, 직급, 급여 순으로 조회한다.

```
select empno, ename, job, sal from emp
where sal > (select sal from emp where empno = 7698);
```

실습 3:

emp 테이블에서 사원번호가 7698 인 사원의 급여보다 적은 사원의 사원번호, 사원명, 직급, 급여 순으로 조회한다.

```
select empno, ename, job, sal from emp
where sal < (select sal from emp where empno = 7698);
```

2) 그룹 함수 적용

단일행을 반환하는 서브쿼리에 그룹 함수를 사용하여 메인쿼리로부터 데이터를 출력할 수 있다.

실습:
emp 테이블에서 부서번호가 20 번인 부서의 최소 급여보다 많은 부서를 조회한다.

```
select deptno, min(sal) from emp
group by deptno
having min(sal) > (select min(sal) from emp where deptno = 20);
```

3) 조회 순번 적용

(1) rownum 컬럼
rownum 컬럼은 슈도(Pseudo) 컬럼으로 mysql 데이터베이스의 limit 키워드와 같은 기능이다.
rownum 컬럼은 쿼리의 결과에 차례대로 숫자 1 부터 순차적으로 번호를 부여한다.
rownum 컬럼은 게시판의 페이징 처리할 때 사용한다.

실습 1:
emp 테이블에서 rownum 슈도 컬럼으로 ename 컬럼을 10 개까지만 조회한다.

```
select rownum, ename from (select * from emp order by empno)
where rownum <= 10;
```

실습 2:
emp 테이블에서 rownum 슈도 컬럼의 별칭으로 ename 컬럼을 10 개까지만 조회한다.

```
select * from (select rownum num, ename from emp order by empno)
where num <= 10;
```

(2) row_number( ) over(order by column…) 문

형식:
row_number( ) over(order by column1, column2, … columnN)

row_number 함수는 번호를 입력하는 함수로 over 함수와 order by 문과 같이 사용한다.
over 함수는 order by 문이나 group by 문의 서브쿼리를 개선하기 위해 나온 함수이다.
over 함수는 집계함수나 분석함수와 함께 사용된다.
row_number( ) over(order by column…) 문은 게시판의 페이징 처리할 때 사용한다.

실습:
emp 테이블에서 row_number( ) over(order by column…) 문으로 ename 컬럼을
10 개까지만 조회한다.

```
select * from (select row_number( ) over(order by empno) as num, ename from
emp)
where num <= 10;
```

2) 다중행 서브쿼리

다중행 서브쿼리는 select 문으로부터 하나 이상의 행을 조회한다.

다중 행 서브쿼리는 다중행 연산자를 사용하며 다중행 연산자는 다음과 같다.

- in 연산자 : 같은 값을 찾는다.
- all 연산자 : 최대값과 최소값을 반환한다.
- any 연산자 : 최소값과 최대값을 반환한다.
- exists 연산자 : 서브쿼리에 값이 있을 때 반환한다.

(1) in 연산자
in 연산자는 2 개 이상의 값을 반환하는 서브쿼리에 대하여 비교 연산자를 비교하여 반환하면 에러가 발생하므로 이런 경우 서브쿼리에서 반환된 목록에 대한 각각의 값을 비교하여 값의 결과가 참일 때 쿼리를 수행한다.
in 연산자는 메인쿼리의 비교 조건에서 서브쿼리의 출력 결과와 하나라도 일치하면 메인쿼리의 where 문으로 조회할 조건을 참으로 반환한다.
in 연산자는 =(이퀄) 연산자로 or 연산자로 연결한 것과 같은 의미이다.

실습:
emp 테이블에서 20 번을 초과한 부서번호의 사원 급여를 조회한다.

```
select ename, sal, deptno from emp
where sal in (select sal from emp where deptno > 20);
```

(2) any 연산자
any 연산자는 비교 연산자와 서브쿼리 사이에 반환된 목록에 대한 각각의 값을 비교하여 값의 결과가 참일 때 쿼리를 수행한다.
any 연산자는 메인쿼리의 비교 조건이 서브쿼리의 조회 결과와 하나 이상만 일치하면 참으로 반환한다.
any 연산자는 서브쿼리의 결과값 중에 어느 하나의 값이라도 만족이 되면 결과값을 반환한다.

실습:
emp 테이블에서 30 번인 부서번호의 사원 중에 최소 급여보다 많은 급여를 받는 사원을 조회한다.

```
select ename, sal, deptno from emp
where sal > any (select sal from emp where deptno = 30);
```

(3) all 연산자
all 연산자는 비교 연산자와 서브쿼리 사이에 반환된 목록에 대한 모든 값을 비교하여 값의 결과가 참일 때 쿼리를 수행한다.
all 연산자는 메인쿼리의 비교 조건이 서브쿼리의 조회 결과와 모든 값이 일치하면 참으로 반환한다.
all 연산자는 서브쿼리의 결과값 중에 모든 결과값이 만족하여야만 결과값을 반환한다.

실습:
emp 테이블에서 30 번인 부서번호의 사원 중에 최대 급여보다 많은 급여를 받는 사원을 조회한다.

```
select ename, sal, deptno from emp
where sal > all (select sal from emp where deptno = 30);
```

(3) exists 연산자
exists 연산자는 서브쿼리에서 적어도 1 개의 행을 반환하면 논리식은 참이고 그렇지 않으면 거짓이 된다.
exists 연산자는 서브쿼리의 결과값이 참이 나오기만 하면 바로 메인쿼리의 결과값을 반환한다.
exists 연산자를 사용하면 서브쿼리의 데이터가 존재하는가를 먼저 따져 존재하는 값들만을 결과로 반환한다.

실습:
emp 테이블의 부서번호가 30 인 조건에 의해서 dept 테이블을 조회한다.

```
select * from dept
where exists (select * from emp where deptno = 30);
```

뷰(View)

1. 뷰의 개요

뷰(view)는 테이블에 대한 가상의 테이블이다.
뷰는 실제 데이터가 저장되는 것은 아니지만 뷰를 통해 데이터를 관리할 수 있다.
뷰는 테이블을 기준으로 생성되며 기본 테이블이 없으면 뷰도 존재할 수 없다.
뷰는 테이블의 전체 데이터 중에서 일부만 접근할 수 있도록 제한한다.

뷰에 대한 수정한 결과는 뷰를 정의한 기본 테이블에서 적용한다.
뷰를 정의한 기본 테이블에서 정의된 무결성 제약조건을 상속받는다.
뷰를 사용하는 이유는 기본 테이블에 대한 보안 기능을 설정해야 하는 경우이거나 복잡하며 자주 사용되는 질의를 더 쉽고 간단하게 사용하기 위해서이다.

뷰의 장점:

- 뷰는 데이터베이스의 선택적인 내용을 보여줄 수 있으므로 데이터베이스에 대한 액세스를 제한한다.
- 복잡한 데이터 질의어를 통해 얻을 수 있는 결과를 간단한 데이터 질의어를 사용한다.
- 데이터 독립성을 허용한다.
- 필요한 데이터만 조회할 수 있으므로 사용자의 편의성을 제공한다.

뷰의 형태:

단순 뷰(simple view): 하나의 기본 테이블에서만 뷰를 생성한다. 단순 뷰에서 실행한 데이터 조작어의 실행 결과는 기본 테이블에 반영한다.

복합 뷰(complex view): 두 개 이상의 기본 테이블에서 뷰를 생성한다. 복합 뷰는 group by 문의 여부에 따라 데이터 조작어를 제한적으로 사용한다.

1. 뷰의 생성

형식

create [or replace] view 뷰명
as
서브쿼리;

create view 문 내에서 서브쿼리를 내장하여 뷰를 생성한다.
서브쿼리는 복합 select 문을 포함할 수 있지만, order by 문은 포함할 수 없다.
뷰 생성 시 컬럼 이름을 명시하지 않으면 기본 테이블의 컬럼 이름을 상속받는다.
함수나 표현식에 의해 정의된 컬럼은 별칭을 사용하여 명시한다.

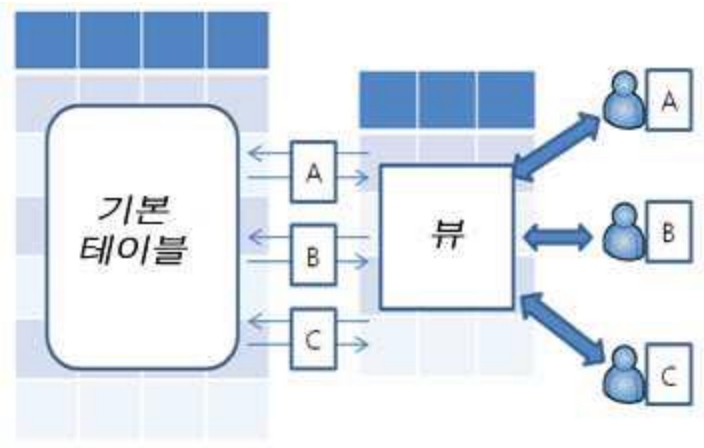
실습:
emp 테이블과 dept 테이블에서 월급에 관련된 내용만 조회할 수 있는 뷰를 생성한다. scott 계정에는 뷰 생성 권한이 없으므로 최고관리자 SYS 계정으로 접속하여 scott 계정에 grant create view to scott 문으로 뷰 생성 권한을 부여한다.

```
-- 뷰를 생성한다.
create or replace view son_emp
as

-- 월급의 총합, 최대값, 최소값을 함수로 구한다.
select e.deptno, d.dname, sum(sal) totsal, max(sal) maxsal, min(sal) minsal
from emp e, dept d
where e.deptno = d.deptno
group by e.deptno, d.dname;

-- 뷰를 조회한다.
select * from son_emp;
```

2. 뷰의 삭제

형식 drop view 뷰명;

drop view 문으로 뷰를 삭제하며 뷰를 정의한 기본 테이블의 구조나 데이터에는 전혀 영향을 주지 않는다.



실습:
생성한 뷰를 삭제한다.

drop view son_emp;

인덱스

1. 인덱스의 개요

인덱스(index)는 포인터를 사용하여 행의 조회를 빠르게 할 수 있는 스키마 객체이다.
인덱스는 테이블 내의 원하는 레코드를 빠르게 찾아갈 수 있도록 만들어진 데이터 구조이다.
인덱스는 명시적 생성하거나 자동으로 생성할 수 있다.
컬럼에 대한 인덱스가 있으면 테이블 전체를 조회한다.
쿼리의 성능을 향상하려면 인덱스 생성을 고려해야 한다.
컬럼이나 컬럼의 집합에 값이 유일하게 하려고 인덱스를 사용할 수 있다.

인덱스의 특징:

- 인덱스는 테이블의 값을 빠르게 액세스하도록 하는 데이터베이스 객체이다.
- 인덱스를 자동으로 사용하고 유지보수 한다.
- 인덱스를 만들면 사용자가 직접 조작할 필요가 없게 된다.
- 인덱스는 테이블과는 독립적이다.
- 인덱스는 언제든지 생성하거나 삭제할 수 있다.
- 인덱스는 다른 인덱스에 영향을 주지 않는다.

인덱스의 형태:
자동 인덱스: primary key 제약조건이나 unique 제약조건에 의해 자동으로 생성되는 인덱스이다.
수동 인덱스: create index 문을 실행해서 만드는 인덱스이다.

인덱스를 생성하는 것이 좋은 컬럼은 null 값이 많이 포함된 컬럼이나 조인 조건에서 자주 사용되는 컬럼이다.
인덱스를 생성하면 오히려 성능이 저하되는 컬럼은 테이블의 크기가 작거나 테이블이 자주 갱신되는 테이블의 컬럼이다.

오라클 인덱스는 B-tree(binary search tree)에 대한 원리를 기반으로 하고 있으며 B-tree 인덱스는 컬럼 안에 독특한 데이터가 많을 때 가장 좋은 효과를 낸다.

B-tree 인덱스의 알고리즘 원리는 주어진 값을 리스트의 중간 지점에 있는 값과 비교하고 만약에 그 값이 더 크면 리스트의 아래쪽 반을 버리고 만약 그 값이 더 작다면 위쪽 반을 버린다.
하나의 값이 발견될 때까지 또는 리스트가 끝날 때까지 그와 같은 작업을 다른 반쪽에도 반복한다.

2. 인덱스의 생성

형식 create [bitmap][unique] index 인덱스명 on 테이블(컬럼)

create index 문으로 인덱스를 생성한다.
on 명령어로 인덱스를 부여할 컬럼을 지정한다.

인덱스의 형태
(1) bitmap 인덱스:
각 컬럼에 대해 적은 개수의 독특한 값이 있을 때 가장 잘 작동한다.
bitmap 인덱스는 B-tree 인덱스가 사용되지 않을 때 성능을 향상한다.
bitmap 인덱스는 테이블이 매우 크거나 수정이 잘 일어나지 않을 때 사용할 수 있다.
질의문이 or 연산자를 포함하는 여러 개의 where 문을 자주 사용할 때 유리하다.

(2) unique 인덱스:
unique 인덱스는 인덱스를 사용한 컬럼의 중복 값들을 포함하지 않고 사용할 수 있는 장점이 있으며 primary key 제약조건이나 unique 제약조건에 의해 자동으로 생성되는 인덱스이다.

(3) non-unique 인덱스
non-unique 인덱스는 인덱스를 사용한 컬럼에 중복 데이터값을 가질 수 있다.

실습:
자동 인덱스로 인덱스를 생성한 테이블의 인덱스와 인덱스 컬럼을 확인한다.

```
select table_name, index_name, column_name from user_ind_columns
where table_name in('EMP');
```

실습:

행에 대한 액세스 속도를 빠르게 하려고 컬럼에 non_unique 인덱스를 생성한다.

```
-- emp 테이블을 복사한다.
create table emp_index
as
select * from emp;
-- 시간 체크를 설정한다.
set timing on;
-- 사원명을 조회한다.
select empno, ename from emp_index
where ename='SMITH';
-- 인덱스를 생성한다.
create index idx_emp on emp_index(ename);
-- 인덱스를 통해서 사원명을 조회한다.
select empno, ename from emp_index
where ename='SMITH';
```

3. 인덱스의 삭제

형식 drop index 인덱스명

drop index 문으로 인덱스를 삭제한다.
인덱스의 구조는 테이블과 독립적이므로 인덱스의 삭제는 테이블의 데이터에는 아무런 영향도 미치지 않는다.
인덱스를 삭제하려면 인덱스의 소유자이거나 인덱스 삭제 권한을 가지고 있어야 한다.
인덱스는 변경할 수가 없고 삭제만 가능하다.

실습:
생성한 인덱스를 삭제한다.

```
-- 인덱스를 삭제한다.
drop index idx_emp;
```