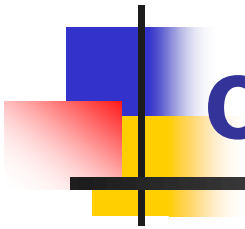
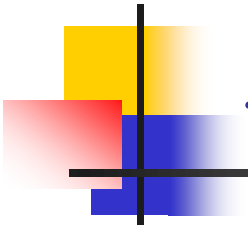


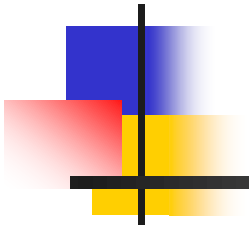
Oracle 8강 –index, view, sequence

양 명 숙
[now4ever7@gmail.com]



목차

- sequence
- index
- view



시퀀스, 인덱스, 뷰

시퀀스 (SEQUENCE)

- 연속적인 숫자를 생성해내는 객체
- 기본 키가 각각의 입력되는 row를 식별할 수 있기만 하면 된다고 할 때, 시퀀스에 의해 생성된 값을 사용
  - 테이블에 있는 기본키 값을 생성하기 위해 사용되는 독립적인 객체
  - 테이블에 종속되지 않음 => 하나의 시퀀스를 여러 개의 테이블에 동시에 사용할 수 있다.

```
CREATE SEQUENCE "스키마명.시퀀스명"
        MINVALUE  -- 시퀀스가 시작되는 최초의 숫자
        MAXVALUE  --시퀀스가 끝나는 최대 숫자
        INCREMENT BY  -- 시퀀스가 증가되는 단위
        START WITH  -- 시퀀스 생성이 시작되는 값
        NOCACHE
        NOORDER
         NOCYCLE;
```

• cache – 오라클에서 시퀀스를 생성하기 위해 미리 값을 할당해 놓기 때문에 시퀀스에 좀 더 빠르게 접근이 가능
 - 동시 사용자가 많을 경우 이 옵션을 사용
• order – 요청되는 순서대로 값을 생성
• cycle – 생성된 시퀀스 값이 최대치 혹은 최소치에 다다랐을 때에 초기값부터 다시 시작할 지 여부

시퀀스 (SEQUENCE)

- NEXTVAL, CURRVAL 의사컬럼
  - NEXTVAL – 바로 다음에 생성될 시퀀스를 가지고 있다
  - CURRVAL – 현재 시퀀스 값을 가지고 있다

```
CREATE SEQUENCE test1_seq
  MINVALUE 1
MAXVALUE 999999999999999999999999999
INCREMENT BY 1
START WITH 207
NOCACHE
NOORDER
NOCYCLE ;

INSERT INTO test1 (id, name)
VALUES (test1_seq.NEXTVAL, '홍길동');
INSERT INTO test1 (id, name)
VALUES (test1_seq.NEXTVAL, '김연아');

SELECT *   FROM test1;
SELECT test1_seq.CURRVAL  FROM DUAL;
SELECT test1_seq.NEXTVAL  FROM DUAL;  --nextval 은 조회할 때마다 값이 증가함
```

시퀀스 (SEQUENCE)

- 시퀀스의 삭제

DROP SEQUENCE “스키마명.시퀀스명”;

- 시퀀스 조회

select * from user_sequences;

예제 – sequence 객체

```
--sequence 객체
create sequence pd2_seq
increment by 1
start with 1
nocache;

insert into pd2(no, pdcode, pdname)
values(pd2_seq.nextval, 'A10', '컴퓨터');
insert into pd2(no, pdcode, pdname)
values(pd2_seq.nextval, 'A11', '키보드');
insert into pd2(no, pdcode, pdname)
values(pd2_seq.nextval, 'A13', '마우스');

select * from pd2;

select pd2_seq.currval from dual;
select pd2_seq.nextval from dual;

select * from user_sequences;
```

```
create sequence dept_seq
increment by 1
start with 100
nocache;

drop sequence dept_seq;
```

인덱스 (INDEX)

- 인덱스
  - 테이블의 데이터를 빨리 찾기 위한 꼬리표
  - 인덱스가 없다면 특정한 값을 찾기 위해 모든 데이터 페이지를 다 뒤져야 함
    – table full scan
  - index seek
    - 인덱스를 사용하는 것이 더 효과적이라면, 오라클은 모든 페이지를 뒤지지 않고 인덱스 페이지를 찾아서 쉽게 데이터를 가져옴
- 인덱스의 생성
  - 테이블에 있는 컬럼을 지정하여 만들 수 있는데 테이블 생성과 동시에 인덱스를 생성할 수는 없다.
  - 인덱스는 테이블과 같이 별도의 데이터베이스 오브젝트로써 테이블과 동등한 레벨의 객체로 존재함
  - 한 테이블에 여러 개의 인덱스를 생성할 수 있음

```
CREATE [UNIQUE] INDEX [스키마명.] 인덱스명
        ON [스키마명.] 테이블명 (컬럼1, [컬럼2, 컬럼3 ...]);
```

인덱스 (INDEX)

사원 테이블

사번	이름	주소	급여
1000	홍길동	서울	400
1001	강감찬	대전	250
1002	일지매	경기	520
1003	나한지	제주	200

IDX_사원_이름 인덱스

Key	ROWID
강감찬	AAASHOAAEAAAACXAAM
나한지	AAASHOAAEAAAACXAAN
일지매	AAASHOAAEAAAACXAAO
홍길동	AAASHOAAEAAAACXAAP

테이블에 있는 인덱스 컬럼의 데이터에 대한 인덱스 정보가 별도로 저장됨
저장된 인덱스를 통해 데이터 조회

인덱스 (INDEX)

```
--인덱스

CREATE UNIQUE INDEX "HR"."EMP_EMAIL_UK" ON "HR"."EMPLOYEES" ("EMAIL"); --단일 인덱스

CREATE INDEX  EMP_DEPARTMENT_IX  ON   EMPLOYEES (DEPARTMENT_ID);

CREATE INDEX  EMP_NAME_IX   ON  EMPLOYEES (LAST_NAME, FIRST_NAME);  --복합인덱스
```

- 인덱스의 삭제

```
DROP INDEX [스키마명.] 인덱스명;
```

- 인덱스 조회

```
select * from user_indexes;
```

인덱스 (INDEX)의 종류

- 유일성 여부에 따라
  - 단일 인덱스 – 인덱스가 한 개의 컬럼에 있을 경우
  - 복합 인덱스 – 인덱스가 두 개 이상의 컬럼에 있을 경우
    - where 절에서 사용되는 컬럼 순으로 인덱스를 구성
- 인덱스 구성 컬럼의 개수에 따라
  - UNIQUE 인덱스 – unique 옵션이 주어진 것
    - 해당 컬럼에 입력되는 값은 유일해야 함
    - unique 키나 기본키를 생성하면 unique 인덱스가 자동으로 생성됨
  - NON-UNIQUE 인덱스 – unique 옵션이 주어지지 않은 것
- 인덱스 생성자에 따라
  - 수동 인덱스 – create index 문을 사용하여 사용자가 생성
  - 자동 인덱스 – unique 키나 기본키 생성에 따라 오라클이 자동으로 생성

인덱스 생성 가이드 라인

- 데이터의 변경이 일어나면 오라클이 인덱스 정보를 자동으로 변경함
  - => 인덱스가 너무 많으면 성능에 문제가 발생함
- 인덱스를 통한 검색
  - 인덱스 정보를 검색하고 나서 검색된 인덱스가 가리키는 데이터를 조회
- 가이드 라인
  - **자주 조회되는 컬럼을** 인덱스 컬럼으로 선택
  - 참조 제약조건(외래키)이 있는 컬럼에 생성한다=>조인 사용할 때에 조인의 속도를 향상시킴
  - 테이블 간 조인에 사용된 컬럼을 인덱스 컬럼으로 선택하면 조인 성능이 향상됨
  - 테이블 전체 row가 적은 경우에는 굳이 인덱스를 만들 필요가 없다
  - 복합인덱스를 구성할 경우 컬럼의 순서는 select 문의 where 절에서 좀 더 자주 사용되는 컬럼을 먼저 오게 한다
  - long, long raw 타입 컬럼은 인덱스로 만들 수 없다.
  - 삽입, 갱신, 삭제가 많이 발생하는 테이블에는 인덱스를 너무 많이 만들지 않도록 한다. <= 데이터의 변경이 발생하면 그에 따른 인덱스 정보도 갱신되기 때문에 성능에 영향을 줄 수 있으므로

예제-index 객체

```
--index
select * from pd2;

create index pdname_idx on pd2(pdname);
create index price_idx on pd2(price);
create unique index email_idx on member2(email);
create index zip_idx on member2(zipcode, address);  --복합 인덱스

select * from user_indexes
where table_name='PD2' or table_name='MEMBER2';

drop index price_idx;
drop index email_idx;
```

VIEW의 개념 및 필요성

- VIEW의 개념
  - VIEW는 테이블에 있는 데이터를 보여주는 형식을 정의하는 SELECT문장의 덩어리라고 할 수 있음
  - VIEW는 **실제로 데이터를 가지고 있지는 않지만** 뷰를 통해 데이터를 조회할 수 있고 또 데이터를 입력 수정 삭제할 수 있으며 다른 테이블과 조인도 할 수 있기 때문에 **가상의 논리적인 테이블**이라고 함
  - 이미 존재하는 하나 혹은 그 이상의 테이블에서 원하는 데이터만 정확히 가져올 수 있도록 미리 원하는 컬럼만 모아 가상적으로 만든 테이블
  - 실제 테이블이 아니라 SQL문을 줄여서 표현한 SQL 문장 덩어리
  - **뷰에는 실제 데이터가 존재하지 않는다**
    - 실제 데이터는 원 테이블에만 존재하고 뷰는 이를 보여주는 창문이라고 생각

VIEW의 개념 및 필요성

- VIEW의 필요성
  - VIEW를 사용하는 가장 큰 목적은 보안성과 사용의 편의성
    - **보안성** : 컬럼 단위로 보안을 주는 것 보다는 사용자들에게 테이블에 액세스 하는 권한을 제거하고 뷰를 통해 테이블에 있는 데이터를 액세스 할 수 있도록 할 수 있음
      - 테이블 전체를 보여주어서는 안 되는 경우, 보여줄 컬럼들만 가져오는 뷰를 만들고 이것을 보여주면 됨
      - 숨기고 싶은 컬럼들을 숨겨줄 수 있다
    - **편의성** : 복잡한 조회 문장을 입력하지 않아도 뷰를 조회하면 원하는 결과를 얻을 수 있기 때문에 보다 편리하게 데이터를 조회할 수 있음
      - 조인과 같은 복잡한 쿼리문장들을 매번 실행할 때마다 작성하지 말고, 뷰로 만들어 수월하게 질의할 수 있다
      - 네트웍 트래픽도 줄일 수 있다

뷰 (VIEW)

- 테이블과 흡사 => sql문을 사용할 때 테이블처럼 사용할 수 있다는 뜻
- 뷰는 실제로 데이터를 저장하고 있지 않다
- 실제로 테이블에 저장된 데이터를 뷰를 통해서 보는 것
- 뷰의 종류
  - Read-Only 뷰 – 조회만 가능한 뷰
  - Updatable 뷰 – insert, update, delete 가능한 뷰
- 기준 테이블의 개수에 따라
  - 단일 뷰
  - 조인 뷰
- WITH CHECK OPTION

CREATE [OR REPLACE] VIEW [스키마이름.] 뷰이름 AS
SELECT 문장;

DROP VIEW [스키마이름.] 뷰이름;

뷰

```
--HR 유저에게 view 생성 권한 주기
grant create view to scott;
--revoke create view from scott;
```

- 예) 다른 사용자가 HR의 emp 테이블 정보를 봐야 한다면 다른 사용자에게 emp 테이블의 select 권한을 주면 됨
- emp 테이블의 sal (월급) 정보는 공개 불가
- => scott 사용자는 emp의 영업부(deptno=30) 사원들의 기본정보(이름, job, 입사일)를 검색할 수 있어야 한다면..
  - 이 조건에 맞는 sql문을 만들어 뷰를 생성
  - 새로 생성된 뷰의 select 권한을 scott 사용자에게 할당

```
--scott 유저에게 뷰의 select 권한 할당하기
grant select on hr.v_emp1 to scott;

--뷰의 select 권한 제거하기
revoke select on ht.v_emp1 from scott;
```

뷰

• 뷰는 테이블처럼 사용하면서도 정의된 쿼리의 결과만 볼 수 있으므로 다른 사용자들에게 데이터의 일부만 공개할 수 있다.

```
SELECT ename, job, hiredate
FROM emp
WHERE deptno = 30 ;

CREATE VIEW v_emp1 AS
SELECT ename, job, hiredate
FROM emp
WHERE deptno = 30;

SELECT  * FROM v_emp1;
DESC v_emp1;

--HR 사용자가 research 부서의 사원정보도 조회해야 한다면

CREATE OR REPLACE VIEW v_emp1 AS
SELECT ename, job, hiredate
FROM emp
WHERE deptno  IN (20, 30);

SELECT  * FROM v_emp1;
```

```
--영업부, research 부에 속하는  사원들 중에 1982년 이전에 입사한 사람의 정보를 조회하려면

--emp 테이블 사용
SELECT ename, job, hiredate
FROM EMP
WHERE deptno IN (20, 30)
   AND hiredate < TO_DATE('1982-01-01');

--v_emp1 뷰 사용
SELECT ename, job, hiredate
FROM v_emp1
 WHERE hiredate < TO_DATE('1982-01-01');
```

user 와 스키마

- user 와 스키마
  - user : 사용자, 오라클 서버에 접속하기 위해 사용하는 것이 user
  - schema : 특정 사용자(user)가 만들어 놓은 모든 object 집합

- 예) scott schema : 오라클 서버안에 scott 계정으로 로그인해서 만들어 놓은 모든 것을 다 모아 둔 것
  - scott user 가 만든 table, index, view, constraint, sequence 등을 통틀어서 scott schema 라고 함

스키마 이름.테이블 이름
scott.emp =>scott 스키마에 있는 emp 테이블, scott 사용자로 로그인해 있기 때문에 scott 생략

스키마(Schema) - 임의의 사용자가 생성한 모든 데이터베이스 객체들을 말하며, 스키마 이름은 그 사용자의 이름과 같다.

VIEW를 통한 데이터 수정

- VIEW를 통한 데이터 수정
  - VIEW를 통하여 데이터를 입력, 수정, 삭제할 수 있음
    - 뷰는 가상의 테이블, 뷰에는 아무 데이터도 없다
    - 실제 데이터는 진짜 테이블에만 존재한다
    - 뷰가 아닌 진짜 테이블에서 데이터를 처리하는 것
  - 하지만 한번에 한 테이블에 기반한 컬럼의 데이터만 입력할 수 있으며 뷰를 만들때 가공된 컬럼에는 데이터가 입력될 수 없음
  - 그리고 뷰에 포함되지 않은 테이블 상의 컬럼들은 NULL을 허용하거나 DEFAULT값이 정의되어 있어야 함
  - 기본적으로 뷰를 만들 때 뷰의 조건을 벗어나는 범위로 데이터를 수정할 수 있으며 이를 허용하지 않고자 할 때는 [WITH CHECK OPTION] 을 사용 한다.

VIEW를 통한 데이터 수정

- Read only 뷰

```
CREATE [OR REPLACE] VIEW [스키마이름.] 뷰이름 AS
        SELECT 문장
        with READ ONLY;
```

- Updatable 뷰

```
CREATE [OR REPLACE] VIEW [스키마이름.] 뷰이름 AS
        SELECT 문장;
```

• 뷰의 조회

```
select * from user_views;
```

VIEW를 통한 데이터 수정

```
CREATE OR REPLACE VIEW v_emp1_read_only AS
SELECT ename, job, hiredate
  FROM emp
WHERE deptno IN (20,30)
WITH READ ONLY;

CREATE OR REPLACE VIEW v_emp1_update AS
SELECT ename, job, hiredate
  FROM emp
WHERE deptno IN (20,30);

UPDATE v_emp1_update
    SET ename = 'SMITH2'
 WHERE ename = 'SMITH' ;

UPDATE v_emp1_read_only
SET ename = 'SMITH2'
WHERE ename = 'SMITH' ;  --에러

INSERT INTO v_emp1_update
VALUES('홍길동','CLERK', SYSDATE); --에러
--[Error] NULL을 ("HR"."EMP"."EMPNO") 안에 삽입할 수 없습니다
```

```
CREATE OR REPLACE VIEW v_emp1_update AS
SELECT empno, ename, job, hiredate
  FROM emp
WHERE deptno IN (20,30);

select * from emp;

INSERT INTO v_emp1_update
VALUES(9999, '홍길동','CLERK', SYSDATE);

DELETE v_emp1_update
WHERE empno = 9999; --삭제 안됨, 뷰의 범위를 벗어나므로 조회도 안됨
```

뷰

```
--
SELECT *
FROM USER_UPDATABLE_COLUMNS
WHERE table_name = 'V_EMP1_UPDATE' ;

--
CREATE OR REPLACE VIEW v_emp1_update AS
SELECT empno, ename, job, hiredate
  FROM emp
WHERE deptno IN (20,30)
WITH CHECK OPTION;

INSERT INTO V_EMP1_UPDATE
VALUES(9997, '손연재', 'CLERK', SYSDATE);  --에러
--[Error]  뷰의 WITH CHECK OPTION의 조건에 위배 됩니다

select * from user_constraints
where table_name='EMP';
```

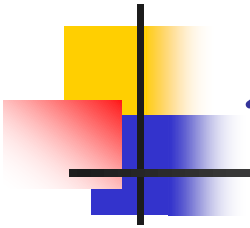
기본적으로 뷰를 만들 때 뷰의 조건을 벗어나는 범위로 데이터를 수정할 수 있으며 이를 허용하지 않고자 할 때는
[WITH CHECK OPTION] 을 사용

뷰

- 조인 뷰

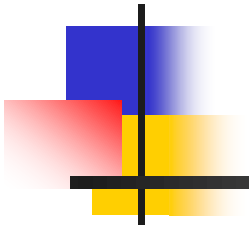
```
CREATE OR REPLACE VIEW v_emp1_dual_update AS
SELECT a.empno, a.ename, a.job, a.hiredate,
         b.deptno, b.dname
  FROM emp a,   dept b
 WHERE a.deptno = b.deptno
   AND b.deptno IN (20, 30);

select * from v_emp1_dual_update;
```



실습

- 장바구니 테이블과 상품 테이블을 이용하여 장바구니 내용을 조회
  - 아이디, 상품명, 가격, 수량, 합계(가격*수량) 컬럼 조회
- 게시판과 한줄 답변 테이블을 이용하여 게시판번호, 작성자, 제목, 내용, 한줄답변 번호, 작성자, 내용, 작성일을 조회(조인)
- 각각 뷰 만들기



분석함수

분석함수

- 분석함수
  - 랭킹, 백분율, 누적합 등 데이터를 분석하는 함수

- 기본형식

```
분석함수(파라미터1, 파라미터2, ...)
OVER ( <partition 절>
         <order by 절> )
```

  - 분석함수 – 사용할 분석함수를 선언한다
    - rank, dens_rank, row_number 등
  - over() 에는 순위를 부여하기 위해 결과 집합을 Partition, Order by 를 통해 분할 및 정렬한다

분석함수

- [1] 순위함수
  - rank, dense_rank, row_number
  - partition 절에 있는 각 행의 순위를 리턴해 주는 함수

```
rank | dense_rank | row_number(expr)
OVER ( <partition  by 컬럼>
          <order by 컬럼> )
```

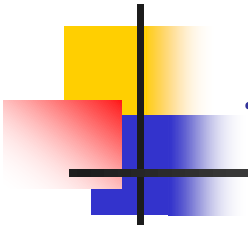
  - 예) 급여가 높은 순서대로 순위를 부여하여 출력하시오

부서번호	사원번호	사원명	기본급	전체등수	부서내등수	전체등수2	부서내등수2	전체등수3	부서내등수3
90	100	Steven King	24000	1	1	1	1	1	1
90	101	Neena Kochhar	17000	2	2	2	2	2	2
90	102	Lex De Haan	17000	2	2	2	2	3	3
80	145	John Russell	14000	4	1	3	1	4	1
80	146	Karen Partners	13500	5	2	4	2	5	2
20	201	Michael Hartstein	13000	6	1	5	1	6	1
100	108	Nancy Greenberg	12008	7	1	6	1	7	1
110	205	Shelley Higgins	12008	7	1	6	1	8	1
80	147	Alberto Errazuriz	12000	9	3	7	3	9	3
80	168	Lisa Ozer	11500	10	4	8	4	10	4
80	174	Ellen Abel	11000	11	5	9	5	11	5
80	148	Gerald Cambrault	11000	11	5	9	5	12	6

순위함수

partition 절에서 department_id를 그룹으로 지정하고 salary 데이터의 높은 급여 순으로 순위를 산출

```
select  department_id as "부서번호",
        employee_id as "사원번호",
        first_name || ' ' || last_name as "사원명",
        salary as "기본급",
        rank() over(order by salary desc) as "전체등수",
        rank() over(partition by department_id
                    order by salary desc) as "부서내등수",
        dense_rank() over(order by salary desc) as "전체등수2",
        dense_rank() over(partition by department_id
                    order by salary desc) as "부서내등수2"  ,
        row_number() over(order by salary desc) as "전체등수3",
        row_number() over(partition by department_id
                    order by salary desc) as "부서내등수3"
 from employees
 --order by "부서번호", "기본급" desc;
 order by "전체등수";
```



순위함수

- rank() 함수
  - 동일한 결과값일 때는 순위가 같고, 다음의 결과값은 동일한 결과값의 건수(n) 만큼 건수(n)를 더하여 순위를 리턴함
  - 올림픽 순위나 프로축구, 야구 등 스포츠 경기 순위를 매길 때 주로 사용됨
- dense_rank() 함수
  - 중복되는 결과값의 건수가 여러 건이라도 다음의 낮은 데이터 순위는 앞 ROW 순위에서 +1로 계산하고 리턴함
- row_number() 함수
  - Partition 내의 분할되어 정렬된 ROW 별 순위를 1부터 순차적으로 적용해서 유일한 수를 리턴함
  - 모든 순위는 개별적인 의미를 가지기 때문에 데이터별로 유일한 순위를 뽑아야 할 때 사용됨

- professor 테이블에서 교수들의 교수번호와 이름, 급여, 급여 순위를 조회하시오

```
select  profno, name, pay,
        rank() over(order by pay) as "RANK",
        rank() over(order by pay desc) as "RANK_desc"
from professor
```

- emp 테이블을 사용하여 사번, 이름, 급여, 부서번호, 부서별 급여순위를 조회하시오
  - partition by – 이 구문 뒤에 그룹핑 할 컬럼을 적어주면 됨

```
select empno, ename, sal,deptno,
   rank() over(partition by deptno order by sal desc) "RANK"
from emp;
```

- emp 테이블을 사용하여 사번, 이름, 급여, 부서번호, 부서 내 job별로 급여순위를 조회하시오

```
select empno, ename, sal,deptno, job,
   rank() over(partition by deptno,job order by sal desc) "RANK"
from emp;
```

- 급여를 많이 받는 상위 5명만 조회, (1~5등까지 조회)

```
/*
 select  profno, name, pay,
        rank() over(order by pay desc) as "RANK"
 from professor
 where rank() over(order by pay desc)<=5;  --error
--rank, dense_rank, row_number 함수는 where 절에 사용 불가
*/

select A.*
 from
 (
    select  profno, name, pay,
           rank() over(order by pay desc) as "RANK"
    from professor
 )A
 where A."RANK"<=5;
```

분석함수

- [2] TOP-N 분석
  - TOP-N 쿼리 – rownum을 이용해 출력건수를 제한
    - 전체 데이터 중 큰 값이나 작은 값 순으로 상위 N개만 출력하고자 할 때 사용됨
  - 예) 최근에 입사한 5명을 순서대로 조회하시오

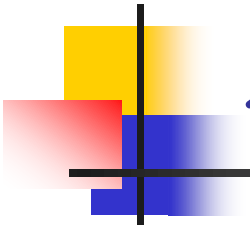
번호	사원번호	이름	직무	입사일자
1	9999	홍길동	CLERK	2015-08-19 21:14:12
2	7876	ADAMS	CLERK	1983-01-12 00:00:00
3	7788	SCOTT	ANALYST	1982-12-09 00:00:00
4	7934	MILLER	CLERK	1982-01-23 00:00:00
5	7900	JAMES	CLERK	1981-12-03 00:00:00

1) rownum 이용

```
select A.*
 from(
   select
      empno "사원번호",
      ename "이름",
      job "직무",
      to_char(hiredate, 'yyyy-mm-dd hh24:mi:ss') as "입사일자"
    from emp
    order by hiredate desc
 ) A
 where rownum<=5;
```

2) row_number() 이용

```
select A.*
 from(
   select
      row_number() over(order by hiredate desc) as "번호",
      empno "사원번호",
      ename "이름",
      job "직무",
      to_char(hiredate, 'yyyy-mm-dd hh24:mi:ss') as "입사일자"
    from emp
 ) A
 where A."번호"<=5;
```



실습

- 1) professor 테이블에서 교수들의 교수번호와 이름, 급여, 급여 순위를 조회하시오
- 2) 1번 예제에서 급여를 가장 많이 받는 3명만 조회하시오
- 3) emp 테이블을 사용하여 사번, 이름, 급여, 부서번호, 부서 내 job별로 급여순위를 조회하시오
- 4) 2번 예제를 뷰로 만드시오.