

View, Index. Sequence

강사 : 강병준

개 요

- 뷰(View)의 개념
- 뷰의 생성
 - 단순 뷰
 - 복합 뷰
- 뷰의 수정과 삭제
- 시퀀스(Sequence)
- 시노님(Synonym)
 - Private 동의어
 - Public 동의어

뷰(View)의 개념

■ 뷰(View) 란?

- 행과 컬럼으로 구성된 가상 테이블(Virtual Table)
- 물리적인 저장 공간과 데이터를 가지지 않고 다른 테이블이나 뷰에서 파생된 논리적인 테이블
- 기본 테이블의 데이터가 변경되면 뷰에도 반영
- 데이터딕셔너리 테이블에 정의된 뷰를 저장한
SELECT : *USER_VIEWS*

뷰(View)의 개념

■ 뷰(View)의 장점

- 뷰를 이용한 기본 테이블의 액세스 제한을 통한 데이터에 대한 보안 기능 제공
- 기본 테이블에 영향을 주지 않고 컬럼명 변경 가능
- 여러 개의 기본 테이블로 정의된 뷰가 하나의 테이블인 것처럼 인식
- 기본 테이블에 대한 복잡한 형태의 질의를 뷰로 정의하여 간단하게 표현 가능

뷰(View)의 종류

■ 단순 뷰(Simple View)

- 하나의 테이블로 구성된 뷰
- INSERT, DELETE, UPDATE와 같은 DML 명령문을 실행하여 기본 테이블의 데이터 조작 가능
- 함수나 그룹 데이터는 사용 가능

■ 복합 뷰(Complex View)

- 하나 이상의 기본 테이블로 구성된 뷰
- DML문을 제한적으로 사용 가능
- 함수나 그룹 데이터는 사용 가능

뷰(View)의 제한조건

- 테이블에 NOT NULL로 만든 컬럼들이 뷰에 다 포함이 되어 있어야 됨
- 그리고 ROWID, ROWNUM, NEXTVAL, CURRVAL등과 같은 가상 컬럼에 대한 참조를 포함하고 있는 뷰에는 어떤 데이터도 Insert할 수 없다.
- WITH READ ONLY 옵션을 설정한 뷰도 데이터를 갱신할 수 없다.
- WITH CHECK OPTION을 설정한 뷰는 뷰의 조건에 해당되는 데이터만 삽입, 삭제, 수정을 할 수 있다.

뷰(View)의 생성

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name  
[(alias1, alias2, .....)]  
AS subquery  
[WITH CHECK OPTION [CONSTRAINT constraint_name]]  
[WITH READ ONLY]
```

- or replace : 기존에 정의된 뷰와 동일한 이름으로 재생성하기 위해 사용
- force : 기본 테이블의 존재 여부에 상관없이 뷰 생성
- noforce : 기본 테이블이 존재할 경우에만 뷰 생성
- alias : 뷰의 컬럼명 지정
- with check option : INSERT나 UPDATE시 서브쿼리의 조건을 만족할 경우에 처리, 무결성 제약 조건명을 명시하지 않을 경우 시스템은 SYS_Cn 형태로 이름 할당
- with read only : 읽기 전용 뷰 생성

뷰(View)의 생성

■ 뷰의 생성 예1

조건>

사원 테이블로 부터 부서 번호가 20인 사원의 단순 뷰 생성

■ 뷰의 생성 확인 : DESC v_emp1

```
select * from v_emp1;
```


뷰(View)의 생성

■ 뷰의 생성 예2

조건>

사원 테이블로 부터 부서 번호가 20인 사원의 사원 번호, 이름, 부서 번호라는 컬럼을 가지는 단순 뷰 생성

```
SQL> CREATE OR REPLACE VIEW v_emp2  
      (사원번호, 이름, 부서번호)  
      AS  
      (SELECT empno, ename, deptno  
       FROM emp  
       WHERE deptno=20);
```

■ 뷰의 생성 확인 : DESC v_emp2

뷰(View)의 생성

컬럼 별칭을 이용하여 뷰를 생성하면, 뷰를 검색 할 때 지정된 컬럼 별칭을 사용해야 한다.

```
SQL> CREATE VIEW SALVU10  
2 AS SELECT EMPNO ID, ENAME NAME, SAL*12 YEAR_SAL  
3 FROM EMP  
4 WHERE DEPTNO = 10;
```

뷰가 생성되었습니다.

```
SQL> SELECT * FROM SALVU10;
```

ID	NAME	YEAR_SAL
7782	CLARK	29400
7839	KING	60000
7934	MILLER	15600

뷰(View)의 생성

두 테이블로부터 값을 출력하는 그룹 함수를 포함하는 복잡한 뷰를 생성합니다. 뷰의 어떤 열이 함수나 표현식에서 유래되었다면 별칭은 필수적입니다.

문제 6) 부서별로 부서명, 최소 급여, 최대 급여, 부서의 평균 급여를 포함하는 DEPT_SUM 뷰를 생성하여라.

```
SQL> CREATE VIEW dept_sum (name, minsal, maxsal, avgsal)
2 AS SELECT d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal)
3 FROM dept d, emp e
4 WHERE d.deptno = e.deptno
5 GROUP BY d.dname;
```

View created.

VIEW에서 DML연산 수행

- 단순 VIEW에서 DML연산을 수행할 수 있습니다.
- VIEW가 다음을 포함 한다면 행을 제거할 수 없습니다.
 - 그룹 함수
 - GROUP BY절
 - DISTINCT키워드
- 다음을 포함한다면 VIEW에서 데이터를 수정할 수 없습니다.
 - 그룹 함수
 - GROUP BY절
 - DISTINCT키워드
 - 표현식으로 정의된 열
 - ROWNUM의사열
- 다음을 포함한다면 VIEW에서 데이터를 추가할 수 없습니다.
 - 그룹 함수
 - GROUP BY절
 - DISTINCT키워드
 - 표현식으로 정의된 열
 - ROWNUM의사열
 - VIEW에 의해 선택되지 않은 NOT NULL열이 기본 테이블에 있을 경우

With Check Option 설정

- 뷰의 조건식을 만족하는 데이터만 INSERT 또는 UPDATE가 가능하도록 하는 옵션

```
SQL> CREATE OR REPLACE VIEW v_Check_Option  
AS  
SELECT empno, ename, deptno  
FROM emp  
WHERE deptno = 10  
WITH CHECK OPTION CONSTRAINT emp_v_cons;
```

```
SQL> INSERT INTO v_Check_Option(empno, ename, deptno)  
VALUES (1005, 'jain', 30);
```

```
INSERT INTO Check_Option(empno, ename, deptno)
```

*

1행에 오류:

ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다

실습하기

WITH CHECK OPTION을 기술하면 뷰를 정의할 때 조건에 사용되어진 칼럼 값을 뷰를 통해서 변경하지 못하도록 하여 혼동을 초래할 만한 일이 생기지 않게 해 봅시다.

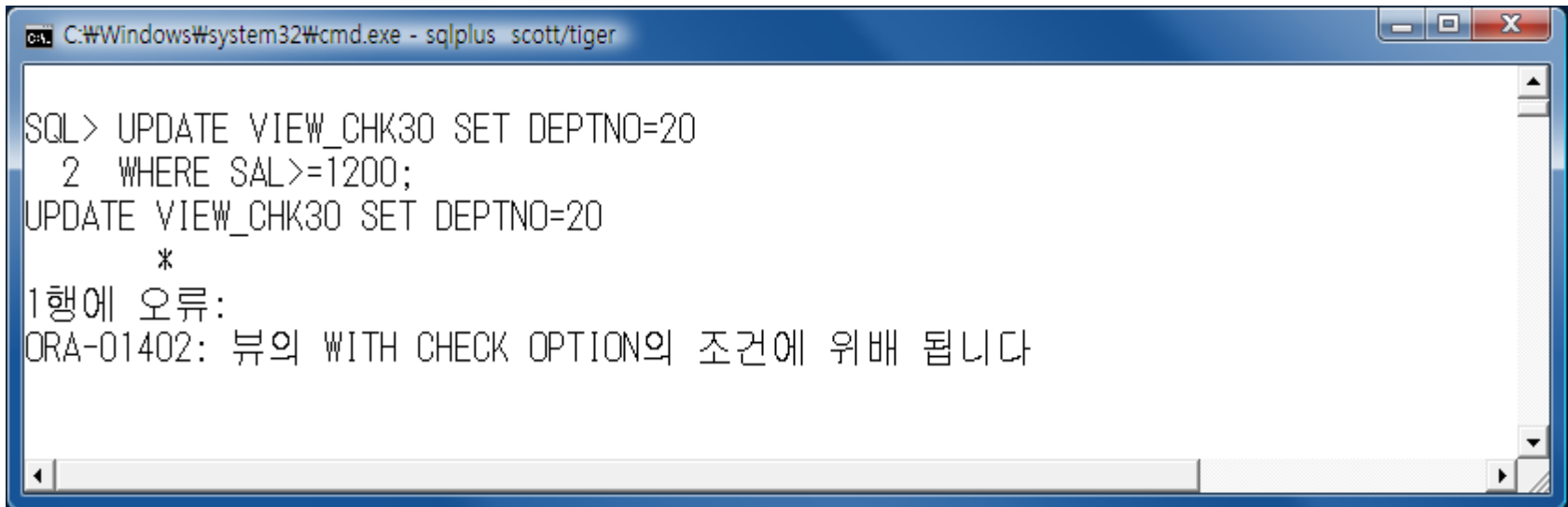
1. 다음과 같이 뷰를 생성합니다.

```
CREATE OR REPLACE VIEW VIEW_CHK30
AS
SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
FROM EMP
WHERE DEPTNO=30 WITH CHECK OPTION;
```

실습하기

- 2. 모든 사원의 부서를 20번 부서로 이동시켜봅시다.

```
UPDATE VIEW_CHK30 SET DEPTNO=20
```

A screenshot of a Windows command prompt window. The title bar reads 'C:\Windows\system32\cmd.exe - sqlplus scott/tiger'. The command prompt shows the following text:

```
SQL> UPDATE VIEW_CHK30 SET DEPTNO=20  
2 WHERE SAL>=1200;  
UPDATE VIEW_CHK30 SET DEPTNO=20  
      *  
1행에 오류:  
ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다
```

- VIEW_CHK30 뷰를 생성할 때 부서번호에 WITH CHECK OPTION을 지정하였기에 이 뷰를 통해서는 부서번호를 변경할 수 없습니다.

With Read Only 설정

- SELECT만 가능한 VIEW를 생성 하는 옵션

```
SQL> CREATE OR REPLACE VIEW v_Read_Only  
AS  
  SELECT empno, ename, deptno  
  FROM emp  
  WHERE deptno = 10  
  WITH READ ONLY
```

view created.

→ 단순히 읽기 만 할 수 있고 데이터는 입력하지 못한다.

오류 발생 예

- 함수 사용시 컬럼 별명을 지정하지 않을 경우

```
SQL> CREATE VIEW v emp3  
      AS SELECT deptno, SUM(sal) SUM, AVG(sal) AVG  
      FROM emp  
      GROUP BY deptno;
```

```
AS SELECT deptno, SUM(sal), AVG(sal)
```

*

ERROR at line 2:

ORA-00998: 이 식은 열의 별명과 함께 지정해야 합니다.

- 서브쿼리에서 함수나 표현식을 사용할 경우 컬럼 별칭을 지정하지 않으면 오류 발생

실습하기

WITH CHECK OPTION을 기술한 VIEW_CHK30 뷰의 커미션을 모두 1000으로 변경해보도록 합시다.

```
UPDATE VIEW_CHK30 SET COMM=1000;
```

WITH CHECK OPTION은 뷰를 설정할 때 조건으로 설정한 컬럼이 아닌 컬럼에 대해서는 변경 가능하므로 커미션이 성공적으로 변경됩니다.

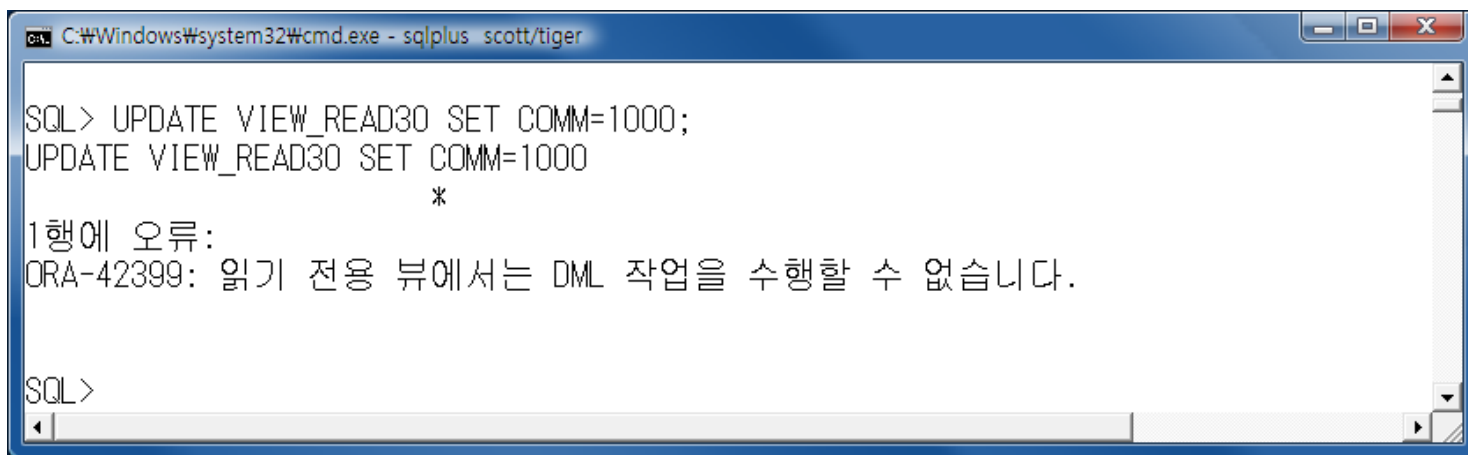
2. WITH READ ONLY 옵션을 지정한 뷰를 정의합니다.

```
CREATE OR REPLACE VIEW VIEW_READ30  
AS  
SELECT EMPNO, ENAME, SAL, COMM, DEPTNO  
FROM EMP  
WHERE DEPTNO=30 WITH READ ONLY;
```

실습하기

3. WITH READ ONLY 옵션을 기술한 VIEW_READ30 뷰의 커미션을 모두 2000으로 변경해보도록 합시다.

```
UPDATE VIEW_READ30 SET COMM=2000;
```

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The window shows the following text:

```
SQL> UPDATE VIEW_READ30 SET COMM=1000;  
UPDATE VIEW_READ30 SET COMM=1000  
          *  
1행에 오류:  
ORA-42399: 읽기 전용 뷰에서는 DML 작업을 수행할 수 없습니다.  
  
SQL>
```

WITH READ ONLY은 뷰를 설정할 때 조건으로 설정한 컬럼이 아닌 컬럼에 대해서도 변경 불가능하므로 커미션이 컬럼 값 역시 변경에 실패합니다.

뷰를 통해서 기본 테이블을 절대 변경할 수 없게 됩니다.

복합 뷰의 생성

- 하나이상의 기본 테이블에서 데이터를 추출하여 만든 가상 테이블

```
SQL> CREATE VIEW v_emp_dept(name, minsal,maxsal,avgsal)
      AS
      SELECT d.dname, MIN(e.sal), MAX(sal), AVG(sal)
      FROM emp e, dept d
      WHERE e.deptno = d.deptno
      GROUP BY d.dname;
```

View created.;

뷰에서의 테이블 업데이트 가능여부 보기

- 무결성 제약 조건을 사용하여 조인된 테이블 사이에 FK 와 PK 관계를 맺은 경우에 한하여 변경 가능
- 변경확인

```
SQL> SELECT column_name, updatable  
FROM user_updatable_columns  
WHERE table_name = 'view 0/름'
```

<i>COLUMN_NAME</i>	<i>UPD</i>
-----	-----
<i>EMPNO</i>	<i>YES</i>
<i>ENAME</i>	<i>YES</i>
<i>JOB</i>	<i>YES</i>
<i>LOC</i>	<i>NO</i>

인라인 뷰

- SQL문장내에 서브쿼리를 사용하여 생성
- inline 뷰는 스키마 object는 아님
- 인라인 뷰: from 절에 테이블 이름이 아닌 서브쿼리를 사용한 것
- ROWNUM: 오라클에서 부여하는 행 번호(pseudo column)
- ROWNUM은 기본적으로 저장된 순서대로 부여되어 있습니다.
- Select 구문을 수행하게 되면 ROWNUM은 select 구문이 실행될 때 다시 만들어집니다.

```
SELECT column_list  
FROM (Subquery) alias  
WHERE condition;
```

조건> 급여가 2000을 초과하는 사원의 평균 급여 출력

```
SQL> SELECT AVG(sal)  
FROM (SELECT sal  
FROM emp  
WHERE sal > 2000) inv_emp;
```

👉 출력 결과

AVG(SAL)

3212.5

From절 상의 서브 쿼리 (InLine View)

- SUBQUERY는 FROM절에서도 사용이 가능
- INLINE VIEW란 FROM절상에 오는 서브쿼리로 VIEW처럼 작용

예제>

급여가 20부서의 평균 급여보다 크고 사원을 관리하는 사원으로서 20부서에 속하지 않은 사원의 정보를 보여주는 SQL문 입니다.

```
SELECT b.empno,b.ename,b.job,b.sal, b.deptno
FROM (SELECT empno
      FROM emp WHERE sal >(SELECT AVG(sal)
                           FROM emp
                           WHERE deptno = 20)) a, emp b
WHERE a.empno = b.empno
      AND b.empno in (select mgr from emp) AND b.deptno !=
20
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7698	BLAKE	MANAGER	2850	30
7782	CLARK	MANAGER	2450	10

From 절상의 서브쿼리 (Inline View)

```
SQL> SELECT  a.ename, a.sal, a.deptno, b.salavg
  2  FROM    emp a, (SELECT  deptno, avg(sal) salavg
  3              FROM      emp
  4              GROUP BY deptno) b
  5  WHERE    a.deptno = b.deptno
  6  AND      a.sal > b.salavg;
```

ENAME	SAL	DEPTNO	SALAVG
KING	5000	10	2916.6667
JONES	2975	20	2175
SCOTT	3000	20	2175
...			

6 rows selected.

실습하기

- 사원 중에서 입사일이 빠른 사람 5명(TOP-5)만을 얻어 오는 질의문을 작성해 봅시다.
- TOP-N을 구하기 위해서는 ROWNUM과 인라인 뷰가 사용됩니다.

1. 다음은 ROWNUM 칼럼 값을 출력하기 위한 쿼리문입니다.

```
SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
FROM EMP;
```

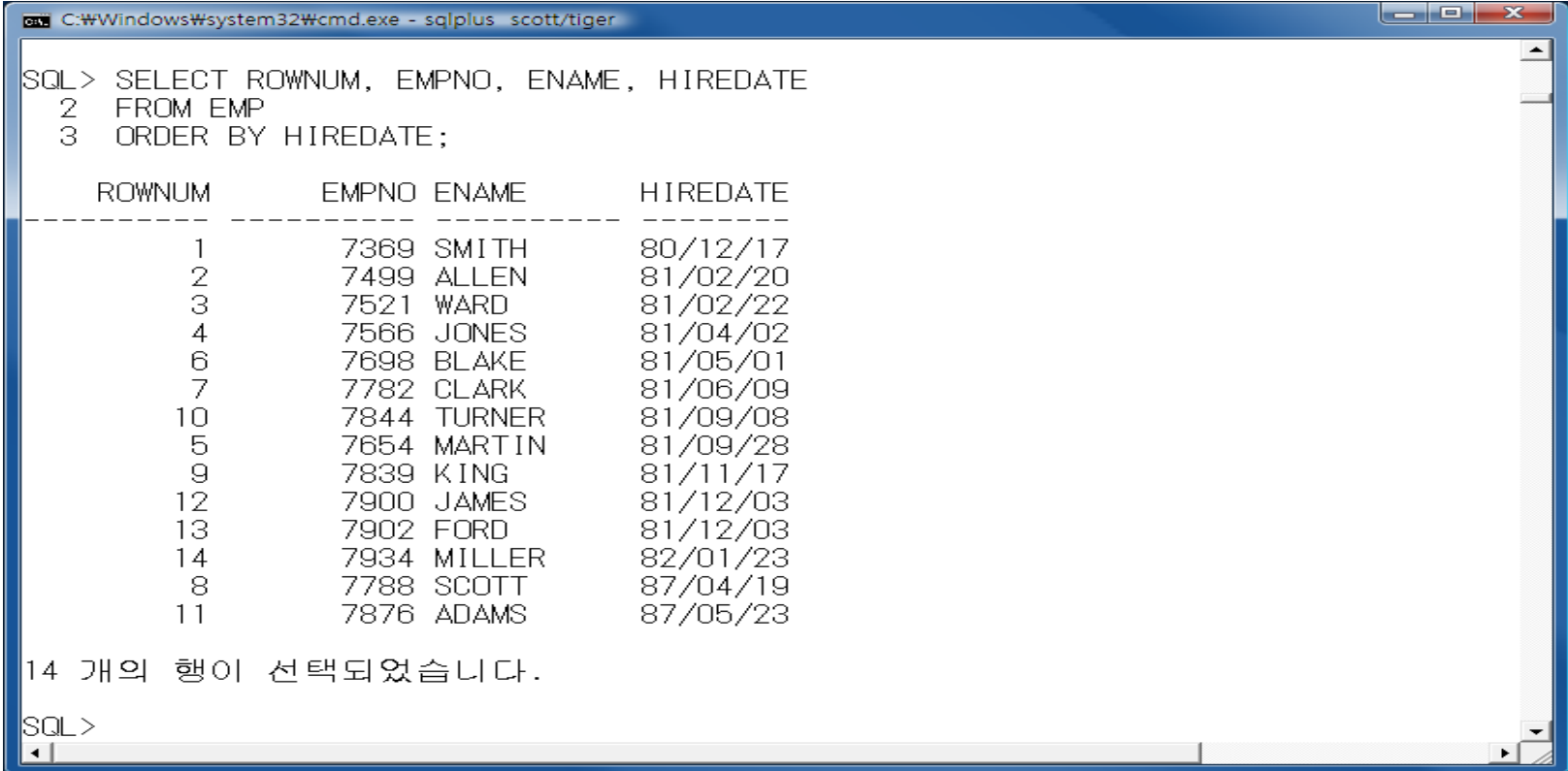
2. 입사일이 빠른 사람 5명만(TOP-N)을 얻어오기 위해서는 일련의 출력 데이터를 일단 임의의 순서로 정렬한 후에 그 중 일부의 데이터만 출력할 수 있도록 해야 하므로 ORDER BY 절을 사용하여 입사일을 기준으로 오름차순 정렬해 봅시다.

```
SELECT EMPNO, ENAME, HIREDATE  
FROM EMP  
ORDER BY HIREDATE;
```

실습하기

3. 이번에는 입사일을 기준으로 오름차순 정렬을 하는 쿼리문에 ROWNUM 칼럼을 출력해 봅시다.

```
SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
FROM EMP  
ORDER BY HIREDATE;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The user has entered the following SQL query:

```
SQL> SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
2 FROM EMP  
3 ORDER BY HIREDATE;
```

The query results are displayed in a table format with four columns: ROWNUM, EMPNO, ENAME, and HIREDATE. The data is sorted by hire date in ascending order. There are 14 rows of data shown.

ROWNUM	EMPNO	ENAME	HIREDATE
1	7369	SMITH	80/12/17
2	7499	ALLEN	81/02/20
3	7521	WARD	81/02/22
4	7566	JONES	81/04/02
6	7698	BLAKE	81/05/01
7	7782	CLARK	81/06/09
10	7844	TURNER	81/09/08
5	7654	MARTIN	81/09/28
9	7839	KING	81/11/17
12	7900	JAMES	81/12/03
13	7902	FORD	81/12/03
14	7934	MILLER	82/01/23
8	7788	SCOTT	87/04/19
11	7876	ADAMS	87/05/23

Below the table, the message "14 개의 행이 선택되었습니다." (14 rows selected) is displayed. The prompt "SQL>" is visible at the bottom of the window.

실습하기

위 결과를 보면 입사일을 기준으로 오름차순 정렬을 하였기에 출력되는 행의 순서는 바뀌더라도 해당 행의 ROWNUM 칼럼 값은 바뀌지 않는다는 것을 알 수 있습니다.

ROWNUM 칼럼은 select 문이 실행될 때 새로 만들어지는 것이므로 이 칼럼의 값을 다른 절에서 사용하려면 새로운 테이블이나 뷰로 새롭게 데이터를 저장해서 사용해야 합니다.

TOP-N은 일련의 출력 데이터를 일단 임의의 순서로 정렬한 후에 그 중 일부의 데이터만 출력할 수 있도록 하여 구합니다.

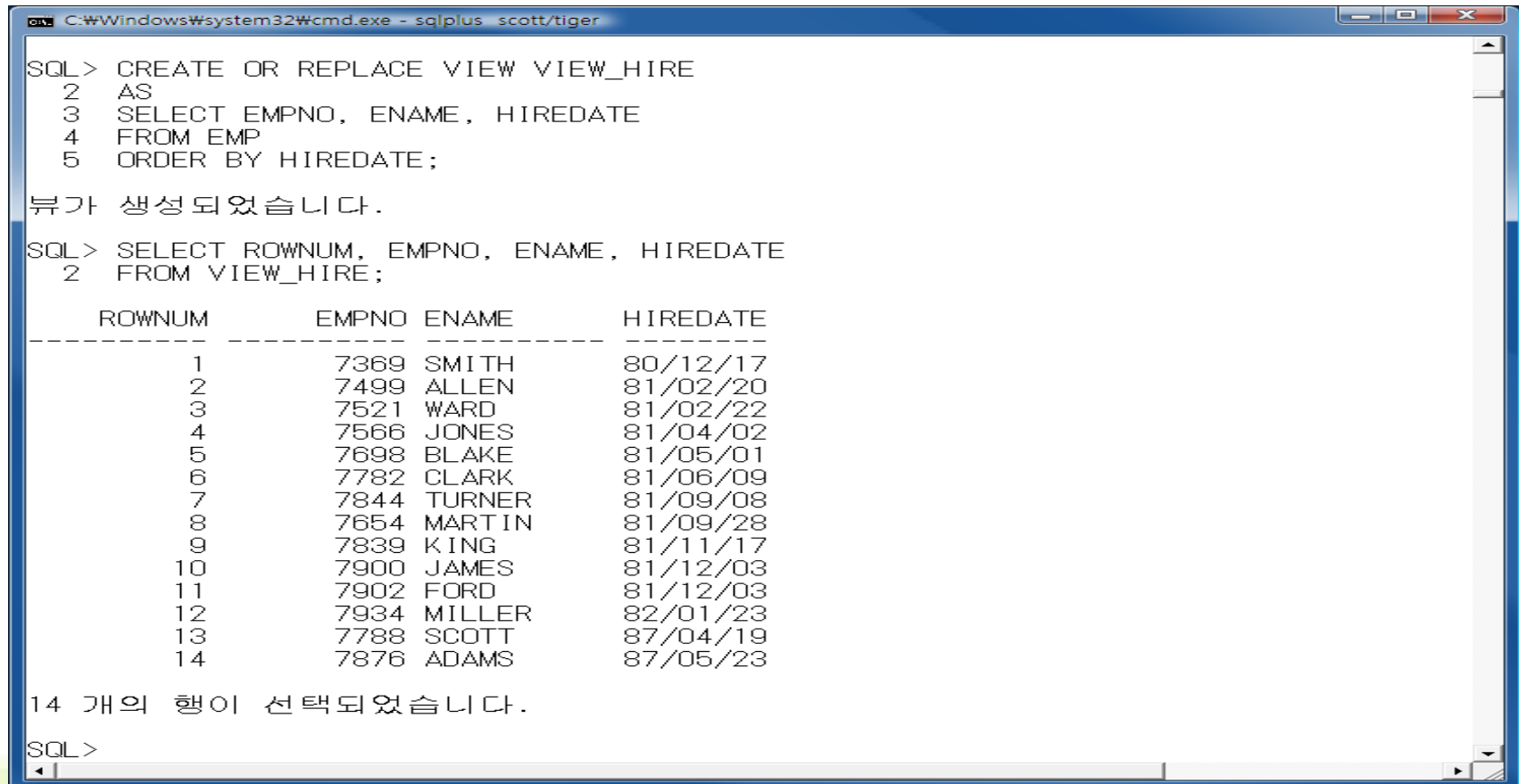
1. 입사일을 기준으로 오름차순 정렬한 쿼리문으로 새로운 뷰를 생성해 봅시다.

```
CREATE OR REPLACE VIEW VIEW_HIRE
AS
SELECT EMPNO, ENAME, HIREDATE
FROM EMP
ORDER BY HIREDATE;
```

실습하기

2. 입사일을 기준으로 오름차순 정렬을 하는 뷰에 ROWNUM 칼럼을 함께 출력해 보시다.

```
SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
FROM VIEW_HIRE;
```



```
C:\Windows\system32\cmd.exe - sqlplus scott/tiger  
SQL> CREATE OR REPLACE VIEW VIEW_HIRE  
2 AS  
3 SELECT EMPNO, ENAME, HIREDATE  
4 FROM EMP  
5 ORDER BY HIREDATE;  
뷰가 생성되었습니다.  
SQL> SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
2 FROM VIEW_HIRE;  

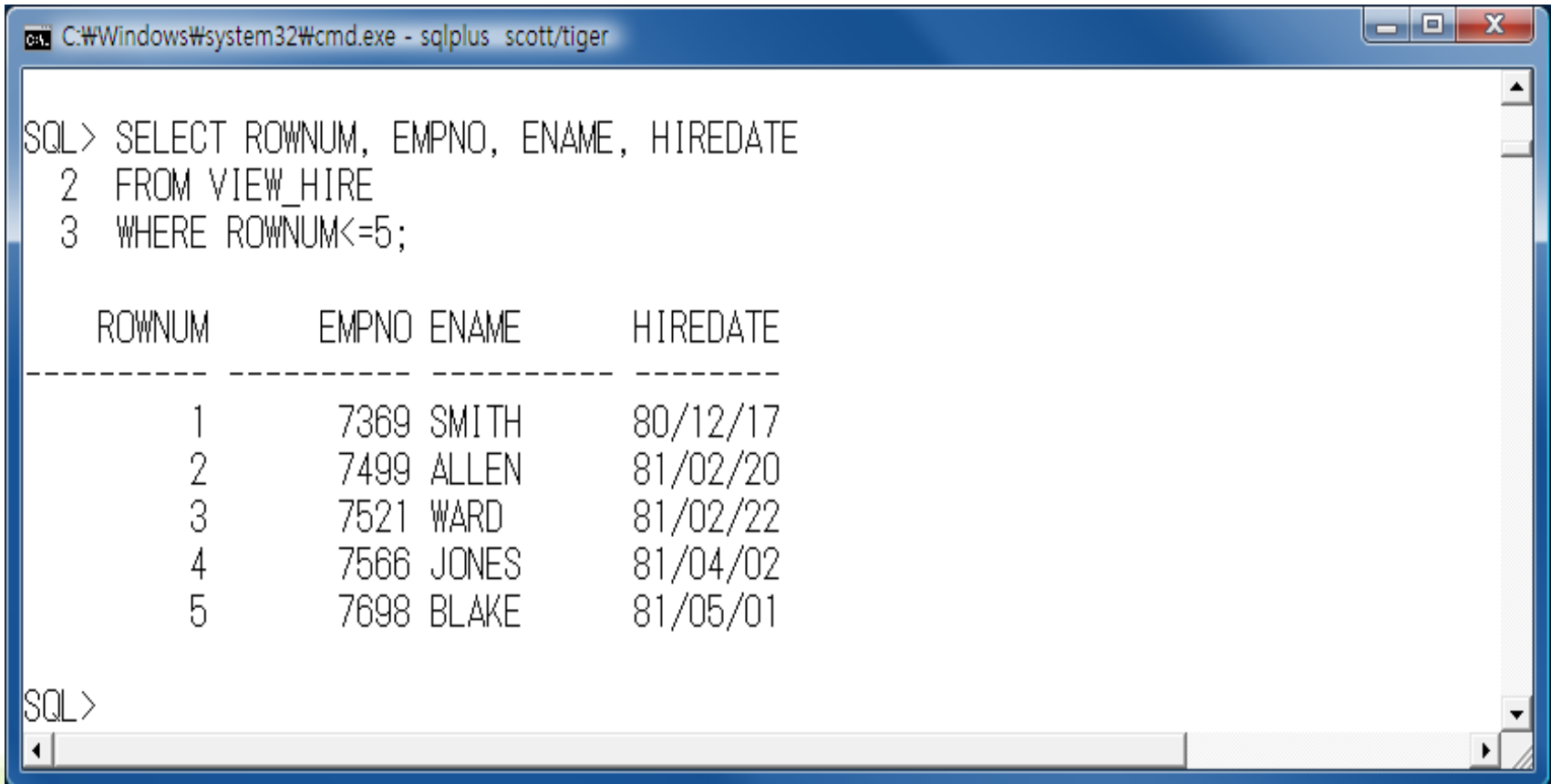

| ROWNUM | EMPNO | ENAME  | HIREDATE |
|--------|-------|--------|----------|
| 1      | 7369  | SMITH  | 80/12/17 |
| 2      | 7499  | ALLEN  | 81/02/20 |
| 3      | 7521  | WARD   | 81/02/22 |
| 4      | 7566  | JONES  | 81/04/02 |
| 5      | 7698  | BLAKE  | 81/05/01 |
| 6      | 7782  | CLARK  | 81/06/09 |
| 7      | 7844  | TURNER | 81/09/08 |
| 8      | 7654  | MARTIN | 81/09/28 |
| 9      | 7839  | KING   | 81/11/17 |
| 10     | 7900  | JAMES  | 81/12/03 |
| 11     | 7902  | FORD   | 81/12/03 |
| 12     | 7934  | MILLER | 82/01/23 |
| 13     | 7788  | SCOTT  | 87/04/19 |
| 14     | 7876  | ADAMS  | 87/05/23 |

  
14 개의 행이 선택되었습니다.  
SQL>
```

실습하기

3. 입사일이 빠른 사람 5명 가져오기

```
SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
FROM VIEW_HIRE  
WHERE ROWNUM<=5;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The user has entered the following SQL query:

```
SQL> SELECT ROWNUM, EMPNO, ENAME, HIREDATE  
2 FROM VIEW_HIRE  
3 WHERE ROWNUM<=5;
```

The query results are displayed in a table format:

ROWNUM	EMPNO	ENAME	HIREDATE
1	7369	SMITH	80/12/17
2	7499	ALLEN	81/02/20
3	7521	WARD	81/02/22
4	7566	JONES	81/04/02
5	7698	BLAKE	81/05/01

The prompt "SQL>" is visible at the bottom left of the window.

뷰에 대한 데이터 디렉터리 정보 확인

- USER_VIEWS 데이터 디렉터리 테이블에 사용자가 정의한 뷰에 대한 정보가 텍스트 형태로 저장

```
SQL> SELECT view_name, text  
FROM user_views;
```

👉 출력 결과

VIEW_NAME	TEXT
-----	-----
V_EMP1	SELECT empno, ename, deptno FROM emp WHERE deptno=20

뷰에서 데이터 조작이 불가능한 경우

- INSERT만 불가능한 경우
 - 뷰의 정의에 포함되지 않는 기본테이블의 컬럼이 NOT NULL 무결성 제약 조건을 가질 경우
- INSERT, UPDATE만 불가능한 경우
 - 표현식으로 정의된 컬럼에 대해서는 UPDATE, INSERT 불가능
 - ROWNUM 와 같은 가상컬럼
- DML명령문 전체가 불가능한 경우
 - 그룹 함수를 포함하는 경우
 - GROUP BY절을 포함하는 경우
 - DISTINCT 키워드를 포함하는 경우
 - WITH READ ONLY 옵션을 가질 때

뷰의 수정1

- 뷰의 수정 시 Alter View 사용
- 단, 뷰의 수정 보다는 재컴파일하거나 유효성을 재검사하기 위해 사용

```
SQL> alter view v_emp1 compile ;
```

```
View altered
```


뷰의 수정2

- CREATE OR REPLACE 명령문 이용
- 뷰 생성시 CREATE 명령문에서 OR REPLACE 옵션 사용
- 실제로는 기존 뷰에 대한 정의를 삭제하고 재정의

```
SQL> CREATE OR REPLACE VIEW v_emp2  
      (사원번호, 이름, 부서번호, 급여)  
      AS  
      SELECT empno, ename, deptno, sal  
      FROM emp  
      WHERE deptno=20;
```

View created.

뷰의 상태 확인

- 뷰의 현재 상태와 유효성 확인을 위해
 - USER_OBJECTS
 - ALL_OBJECTS 사용

```
SQL> select object_name, status  
        from user_objects  
        where object_name = 'V_EMP1';
```

Top-N 구문

- TOP-N 질의는 컬럼의 값중 n 개의 가장 큰 값 또는 작은 값을 질의하는 것

```
SELECT [column_list], ROWNUM  
FROM  
    (SELECT [column_list] FROM table  
    ORDER BY Top-N_column)  
WHERE ROWNUM <= N
```

■ 사용방법

- 데이터를 정렬하는 질의를 서브쿼리나 inline 뷰로 생성

■ 필수 요소

- **Rownum** (Row순서를 나타내는 가상컬럼)
- **From절**상의 **SubQuery** (**Inline View**)
- **SubQuery** 내에 **Order by절**
- 사용가능 비교연산자 : <, <=

Top-N Example

Example)

```
SQL> SELECT ROWNUM as RANK, ename, sal
      FROM (SELECT ename, sal
            FROM EMP
            ORDER BY sal DESC)
      WHERE ROWNUM <= 3;
```

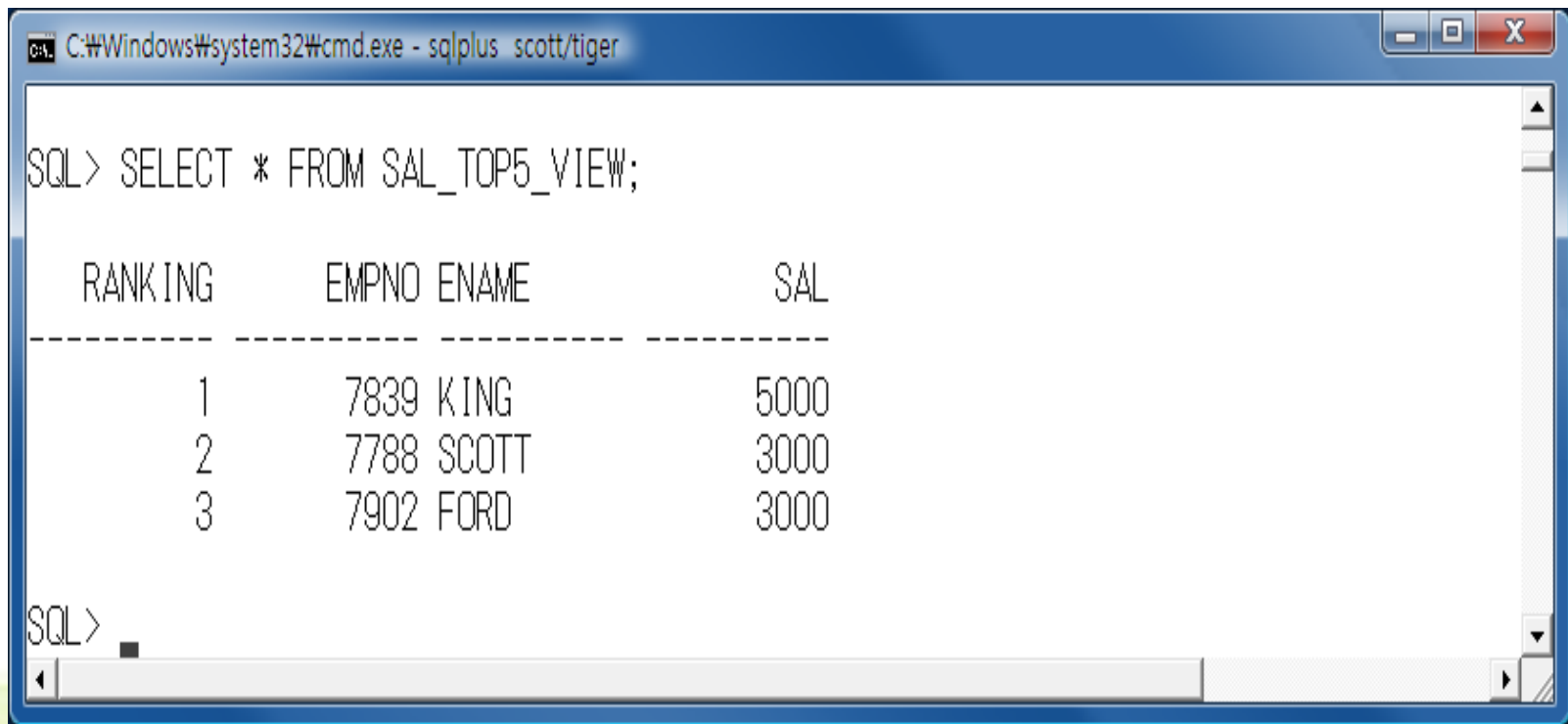
☞ 출력 결과

RANK	ENAME	SAL
-----	-----	-----
1	KING	5000
2	SCOTT	3000
3	FORD	3000

3개의 행들이 선택되었습니다.

연습문제

1. EMP 테이블에서 사원 번호(empno), 이름(ename), 업무(job)를 포함하는 EMP_VIEW VIEW를 생성하여라.
2. 1번에서 생성한 VIEW를 이용하여 10번 부서의 자료만 조회하여라.
3. EMP 테이블과 인라인 뷰를 사용하여 급여를 많이 받는 순서대로 3명만 출력하는 뷰(SAL_TOP5_VIEW)를 작성하시오.



The screenshot shows a SQL*Plus window with the title bar "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The command prompt shows the execution of the query "SQL> SELECT * FROM SAL_TOP5_VIEW;". The result is displayed as a table with four columns: RANKING, EMPNO, ENAME, and SAL. The data shows the top 3 employees by salary: KING (5000), SCOTT (3000), and FORD (3000).

```
SQL> SELECT * FROM SAL_TOP5_VIEW;
```

RANKING	EMPNO	ENAME	SAL
1	7839	KING	5000
2	7788	SCOTT	3000
3	7902	FORD	3000

```
SQL>
```

Tip-N 절의 Rank() 함수 적용

```
SELECT *  
FROM (SELECT EMPNO, ENAME, SAL,  
            RANK() OVER( ORDER BY SAL DESC) AS RANK  
      FROM EMP)  
WHERE RANK <= 3;
```

EMPNO	ENAME	SAL	RANK
7839	KING	5000	1
7788	SCOTT	3000	2
7902	FORD	3000	2
7566	JONES	2975	4
7698	BLAKE	2850	5
7782	CLARK	2450	6
7499	ALLEN	1600	7
7844	TURNER	1500	8
7934	MILLER	1300	9
7521	WARD	1250	10
7654	MARTIN	1250	10

Top-N 절의 Analyze Function 적용

```
SELECT EMPNO, ENAME, SAL,  
       RANK() OVER(ORDER BY SAL DESC) AS RANK,  
       DENSE_RANK() OVER(ORDER BY SAL DESC) AS DENSE_RANK,  
       ROW_NUMBER() OVER(ORDER BY SAL DESC) AS ROW_NUMBER  
FROM EMP;
```

EMPNO	ENAME	SAL	RANK	DENSE_RANK	ROW_NUMBER
7839	KING	5000	1	1	1
7788	SCOTT	3000	2	2	2
7902	FORD	3000	2	2	3
7566	JONES	2975	4	3	4
7698	BLAKE	2850	5	4	5
7782	CLARK	2450	6	5	6
7499	ALLEN	1600	7	6	7
7844	TURNER	1500	8	7	8
7934	MILLER	1300	9	8	9
7521	WARD	1250	10	9	10
7654	MARTIN	1250	10	9	11
7876	ADAMS	1100	12	10	12
7900	JAMES	950	13	11	13
7369	SMITH	800	14	12	14

Top-N 구문 2

예시) 6~10 사이의 급여 랭킹을 구하시오.

WITH절

- 여러 개의 subquery가 하나의 mainquery에서 사용될 때 생기는 복잡성을 보다 간결하게 정의하여 사용함으로써 발생할 수 있는 성능저하 현상을 방지할 수 있음.

<statement>

With

```
[inline_view 이름1] as (select col1
                        from table1
                        where col1),
[inline_view 이름2] as (select col2
                        from table2
                        where col2),
select * from inline_view 이름1
Where col1 > (select col1
              from inline_view 이름2
              where col1);
```

WITH절

- WITH절에 포함된 서브 쿼리를 반환 할 수 있다

조건>

회사의 연간 급여 예산의 1/3이상을 포함하는 부서들의 급여 총합은 ?

```
SQL> WITH summary as
      (SELECT dname, sum(sal) dept_total
       FROM emp, dept
       WHERE emp.deptno = dept.deptno
       GROUP BY dname)
      SELECT dname, dept_total
      FROM summary
      WHERE dept_total >
        (select sum(dept_total) * 1/3
         from summary)
      ORDER BY dept_total desc;
```

DNAME	DEPT_TOTAL
RESEARCH	10875

WITH절

- 아래 쿼리를 작성하고 어떤 결과를 요구한 것인지 말하십시오.

With

```
dept_costs as (select d.dname, sum(e.sal) as dept_total  
                  from emp e, dept d  
                  where e.deptno=d.deptno  
                  group by d.dname),
```

```
avg_cost as (select sum(dept_total) / count(*) as  
                  dept_avg from dept_costs)
```

```
select * from dept_costs  
where dept_total > (select dept_avg from avg_cost)  
order by dname;
```

DNAME	DEPT_TOTAL
-----	-----
RESEARCH	10875

연 습 문 제

1. EMP 테이블에서 사원 번호, 이름, 업무, 부서코드를 포함하는 EMP_VIEW VIEW를 생성하여라.
2. 1번에서 생성한 VIEW를 이용하여 10번 부서의 자료만 조회하여라.
3. EMP 테이블과 DEPT 테이블을 이용하여 이름, 업무, 급여, 부서명, 위치를 포함하는 EMP_DEPT_NAME이라는 VIEW를 생성하여라.
4. VIEW 생성시 WITH READ ONLY OPTION에 대하여 설명하여라.
5. VIEW 생성시 WITH CHECK OPTION에 대하여 설명하여라.
6. VIEW를 이용하여 자료를 수정할 수 있는 경우와 없는 경우에 대하여 설명하여라.

1. 부서명과 사원명을 출력하는 뷰 DNAME_ENAME_VU를 작성하시오.
2. 사원명과 사수명을 출력하는 뷰 WORKER_MANAGER_VU를 작성하시오.
3. 사원 테이블에서 사번, 사원명, 입사일을 입사일이 늦은 사원 순으로 정렬하시오.
4. 사원 테이블에서 사번, 사원명, 입사일을 입사일이 늦은 사원 5명을 출력하시오.
5. 사원 테이블에서 사번, 사원명, 입사일을 입사일이 6번째로 늦은 사원부터 10번째 사원까지 출력하시오.

뷰의 삭제

- DROP VIEW 명령문을 사용하여 뷰 삭제
- 뷰는 기본 테이블에서 파생된 가상 테이블이므로 뷰의 삭제는 데이터베이스에 저장된 뷰 정의를 삭제하는 것
- 삭제된 뷰가 정의된 기본 테이블의 구조나 데이터는 영향을 받지 않음

```
SQL> DROP VIEW view_name
```

VIEW_SAL을 삭제합니다.
DROP VIEW VIEW_SAL;

OR REPLACE 옵션

- CREATE OR REPLACE VIEW 를 사용하면 존재하지 않은 뷰이면 새로운 뷰를 생성하고 기존에 존재하는 뷰이면 그 내용을 변경합니다.
- 이전에 작성한 EMP_VIEW30 뷰는 “EMPNO, ENAME, SAL, DEPTNO” 4개의 컬럼을 출력하는 형태였는데 커미션 칼럼을 추가로 출력할 수 있도록 하기 위해서 뷰의 구조를 변경합니다.

```
CREATE OR REPLACE VIEW EMP_VIEW30
AS
SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
FROM EMP_COPY
WHERE DEPTNO=30;
```

FORCE 옵션

- 뷰를 생성하는 경우에 일반적으로 기본 테이블이 존재한다는 가정 하에서 쿼리문을 작성합니다.
- 극히 드물기는 하지만, 기본 테이블이 존재하지 않는 경우에도 뷰를 생성해야 할 경우가 있습니다. 이럴 경우에 사용하는 것이 FORCE 옵션입니다.
- FORCE 옵션과 반대로 동작하는 것으로서 NOFORCE 옵션이 있습니다.
- NOFORCE 옵션은 반드시 기본 테이블이 존재해야 할 경우에만 뷰가 생성됩니다.
- 지금까지 뷰를 생성하면서 FORCE/NOFORCE 옵션을 지정하지 않았습니다. 이렇게 특별한 설정이 없으면 디폴트로 NOFORCE 옵션이 지정된 것이므로 간주합니다.

실습하기

FORCE/NOFORCE 옵션이 어떤 역할을 하는지 살펴보기 위해서 존재하지 않는 테이블인 EMPLOYEES를 사용하여 뷰를 생성하도록 해봅시다.

존재하지 않는 EMPLOYEES를 기본 테이블로 하여 뷰를 생성하게 되면 다음과 같이 오류가 발생합니다.

```
CREATE OR REPLACE VIEW EMPLOYEES_VIEW  
AS  
SELECT EMPNO, ENAME, DEPTNO  
FROM EMPLOYEES  
WHERE DEPTNO=30;
```

특별한 설정이 없으면 NOFORCE 옵션이 지정된 것이므로 반드시 존재하는 기본 테이블을 이용한 쿼리문으로 뷰를 생성해야 합니다.

실습하기

2. 기본 테이블이 존재하기 않는 경우에도 뷰를 생성하기 위해서 FORCE 옵션이 적용해 봅시다.

```
CREATE OR REPLACE FORCE VIEW NOTABLE_VIEW  
AS  
SELECT EMPNO, ENAME, DEPTNO  
FROM EMPLOYEES  
WHERE DEPTNO=30;
```

존재하지 않는 테이블을 기본 테이블로 지정했다는 경고 메시지는 출력되지만, USER_VIEWS의 내용을 살펴보면 뷰가 생성된 것을 확인할 수 있습니다.

실습하기

- 3. FORCE의 반대 기능을 가진 옵션은 NOFORCE입니다.

```
CREATE OR REPLACE NOFORCE EXISTTABLE_VIEW  
AS  
SELECT EMPNO, ENAME, DEPTNO  
FROM EMPLOYEES  
WHERE DEPTNO=30;
```

- NOFORCE 옵션을 사용하면 뷰를 생성할 때 존재하지 않는 테이블을 기본 테이블로 지정하면 오류 메시지와 함께 뷰가 생성되지 않습니다.

시퀀스(Sequence)의 정의

- 순차적인 번호를 자동으로 생성하는 객체로 테이블과 독립적으로 생성 및 저장 가능
- 시퀀스에서 생성되는 번호는 유일하기 때문에 기본 테이블에서 인조 **Primary Key** 생성시 주로 사용
- 여러 테이블에 의해 공유 가능
- 시퀀스는 테이블과 관계없이 생성, 저장, 오라클 내부 루틴에 의해 발생되고 증가, 감소됨

시퀀스의 생성

- increment by n

: 시퀀스 번호의 증가치로 기본값은 1.

일반적으로 1과 -1을 사용

- start with n

: 시퀀스 시작 번호 지정. 기본값은 1

- maxvalue n

: 생성될 수 있는 최대 시퀀스 값

- nomaxvalue : 시퀀스가 증가할 경우 최대값은 , 감소할 경우 -1

- minvalue n : 생성될 수 있는 최소값

- nominvalue : 시퀀스가 증가할 경우 최소값은 1, 감소할 경우

- cycle | nocycle :

- maxvalue 또는 minvalue에 도달한 후 계속해서 순환적으로 시퀀스 번호를 생성할 것인지 여부 지정
- 기본키에서는 유일성을 보장해야 하므로 nocycle로 지정

- cache n | nocache : 메모리에 캐쉬할 시퀀스 개수 지정, 기본값은 20개

```
CREATE SEQUENCE sequence_name  
[INCREMENT BY n  
[START WITH n  
[MAXVALUE n | NOMAXVALUE]  
[MINVALUE n | NOMINVALUE]  
[CYCLE | NOCYCLE]  
[CACHE n | NOCACHE];
```

시퀀스 생성

```
SQL> create sequence test_seq;
```

```
SQL> create sequence test2_seq  
2 start with 10 increment by 5  
3 minvalue 10 maxvalue 20  
4 cycle cache 2 order;
```

```
SQL> create sequence test3_seq  
2 start with 10 increment by -1  
3 minvalue 1 maxvalue 10  
4 cycle cache 5;
```

시퀀스 생성 확인

- 데이터 저장소(data dictionary)에서 시퀀스(sequences) 정보 모으기

```
SQL> select * from user_sequences;
```

SEQ_NAME	MIN_VAL	MAX_VAL	INC_BY	C	O	CA_SIZE	LAST_NUM
-----	-----	-----	-----	-	-	-----	-----
TEST_SEQ	1	1.0E+27	1	N	N	20	1
TEST2_SEQ	10	20	5	Y	Y	2	10
TEST3_SEQ	1	10	-1	Y	N	5	10

CURRVAL & NEXTVAL

- CURRVAL은 시퀀스의 현재 값
- NEXTVAL은 시퀀스의 다음 값
- 새로운 세션 시작시 NEXTVAL 값을 먼저 생성한 후에 CURRVAL 값 생성 가능
- 제한 사항
 - 뷰를 정의하는 SELECT 명령문에는 사용 불가능
 - DISTINCT 키워드와 함께 사용 불가능
 - GROUP BY, HAVING, ORDER BY절과 함께 사용 불가능
 - 서브쿼리문에 사용 불가능
 - 테이블 생성이나 변경시 컬럼의 기본값 표현식에서는 사용 불가능

CURRVAL & NEXTVAL

- 사용 가능한 절
 - 서브쿼리의 일부가 아닌 SELECT문장의 SELECT 리스트
 - INSERT문장에서 서브쿼리 SELECT 리스트
 - INSERT문장의 VALUES절
 - UPDATE문장의 SET절

<시퀀스 값 확인>

sequence_name.CURRVAL

sequence_name.NEXTVAL

시퀀스 사용

- 시퀀스(sequences) 사용(use)
 - Nextval 을 사용한 초기화가 필요하다.

```
SQL> select test_seq.currval from dual;
```

```
ORA-08002 : sequence test_seq.currval is not yet  
defined in this session
```

```
SQL> select test_seq.nextval, test_seq.currval  
2 from dual;
```

NEXTVAL	CURRVAL
-----	-----
1	1

시퀀스 응용1

- 시퀀스(sequences)를 사용(use)한 기본키(primary key)

```
SQL> create table order_status2 (  
2     id number  
        constraint order_status2_pk primary key,  
3     status varchar2(10),  
4     last_modified date default sysdate  
5 );
```

```
SQL> create sequence order_status2_seq nocache;
```

- 기본키에 사용하는 시퀀스는 데이터베이스 다운 시 캐시값의 손실로 인한 숫자의 갭 발생을 방지하기 위해 **nocache** 옵션을 사용한다.

시퀀스 응용2

```
SQL> insert into order_status2 (id, status)
2 values (
3 order_status2_seq.nextval, 'PLACED'
4 );
```

```
SQL> insert into order_status2 (id, status)
2 values (
3 order_status2_seq.nextval, 'PENDING'
4 );
```

```
SQL> select * from order_status2;
```

ID	STATUS	LAST_MODIFIED
1	PLACED	01-JAN-06
2	PENDING	01-FEB-06

시퀀스의 정의 변경

- **ALTER SEQUENCE sequence** 명령문 사용
- 해당 시퀀스의 소유자이거나 **ALTER SEQUENCE** 권한 필요
- 시퀀스 정의 변경시 새로 생성되는 시퀀스 번호에 대해서만
영향
- **START WITH**절은 변경 불가

```
ALTER SEQUENCE sequence  
[INCREMENT BY n]  
[MAXVALUE n | NOMAXVALUE]  
[MINVALUE n | NOMINVALUE]  
[CYCLE | NOCYCLE]  
[CACHE n | NOCACHE];
```

시퀀스 수정

■ 시퀀스(sequences) 수정(modify)

```
SQL> alter sequence test_seq  
2 increment by 2;
```

```
SQL> select test_seq.currval from dual;
```

```
CURRVAL  
-----  
2
```

```
SQL> select test_seq.nextval from dual;
```

```
NEXTVAL  
-----  
4
```

- **DROP SEQUENCE sequence** 명령문 사용
- 해당 시퀀스의 소유자이거나 **DROP ANY SEQUENCE** 권한 필요

```
DROP SEQUENCE sequence;
```

시퀀스 삭제

■ 시퀀스(sequences) 삭제(drop)

```
SQL> drop sequence test3_seq;
```

```
SQL> select * from user_sequences;
```

SEQ_NAME	MIN_VAL	MAX_VAL	INC_BY	C	O	CA_SIZE	LAST_NUM
TEST_SEQ	1	1.0E+27	1	N	N	20	1
TEST2_SEQ	10	20	5	Y	Y	2	10

실습하기

- 사원 번호를 생성하는 시퀀스 객체를 생성하여 이를 기본 키인 사원 번호에 사용하여 사용자가 새로운 사원을 추가할 때마다 유일한 사원번호를 INSERT 해야 하는 번거로움을 줄입니다.
- 1. 시작 값이 1이고 1씩 증가하고, 최댓값이 100000이 되는 시퀀스 EMP_SEQ 생성합니다.

```
CREATE SEQUENCE EMP_SEQ  
START WITH 1  
INCREMENT BY 1  
MAXVALUE 100000 ;
```

실습하기

2. 이번에는 생성된 시퀀스를 사용하기 위해서 사원 번호를 기본 키로 설정하여 EMP01란 이름으로 새롭게 생성합니다.

```
DROP TABLE EMP01;
```

```
CREATE TABLE EMP01(  
  EMPNO NUMBER(4) PRIMARY KEY,  
  ENAME VARCHAR(10),  
  HIREDATE DATE  
);
```

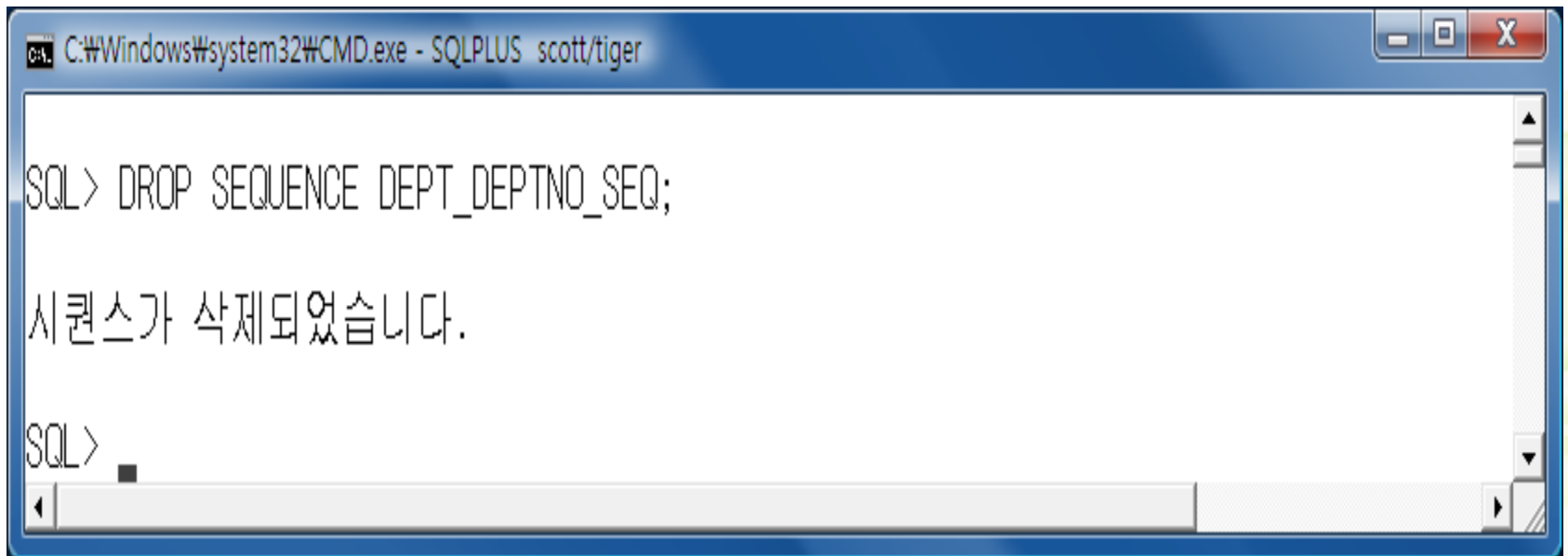
3. 사원 번호를 저장하는 EMPNO 컬럼은 기본 키로 설정하였으므로 중복된 값을 가질 수 없습니다. 다음은 생성한 EMP_SEQ 시퀀스로부터 사원번호를 자동으로 할당 받아 데이터를 추가하는 문장입니다.

```
INSERT INTO EMP01  
VALUES(EMP_SEQ.NEXTVAL, 'JULIA' , SYSDATE);
```

실습하기

- DROP SEQUENCE문으로 시퀀스를 제거해 봅시다.

```
DROP SEQUENCE DEPT_DEPTNO_SEQ;
```



```
C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger

SQL> DROP SEQUENCE DEPT_DEPTNO_SEQ;

시퀀스가 삭제되었습니다.

SQL>
```

연 습 문 제

1. 초기값1부터 최대값999,999까지 1씩 증가하는 TEST_SEQ SEQUENCE를 생성하여라.
2. 현재 SESSION을 이루고 있는 사용자가 사용할 수 있는 SRQUENCE를 조회하여라.
3. CURRVAL과 NEXTVAL을 설명하여라.
4. CACHE와 NOCACHE의 차이점을 설명하여라.
5. CYCLE와 NOCYCLE의 차이점을 설명하여라.
6. 1번에서 생성한 SRQUENCE를 삭제하여라

연 습 문 제

- 사원 번호를 생성하는 시퀀스 객체를 생성하여 이를 기본 키인 사원 번호에 사용하여 사용자가 새로운 사원을 추가할 때마다 유일한 사원번호를 INSERT 해야 하는 번거로움을 줄입니다.
- 1. 시작 값이 1이고 1씩 증가하고, 최댓값이 100000이 되는 시퀀스 EMP_SEQ 생성합니다.

```
CREATE SEQUENCE EMP_SEQ  
START WITH 1  
INCREMENT BY 1  
MAXVALUE 100000 ;
```

연 습 문 제

2. 이번에는 생성된 시퀀스를 사용하기 위해서 사원 번호를 기본 키로 설정하여 EMP01란 이름으로 새롭게 생성합니다.

```
DROP TABLE EMP01;
```

```
CREATE TABLE EMP01(  
EMPNO NUMBER(4) PRIMARY KEY,  
ENAME VARCHAR(10),  
HIREDATE DATE  
);
```

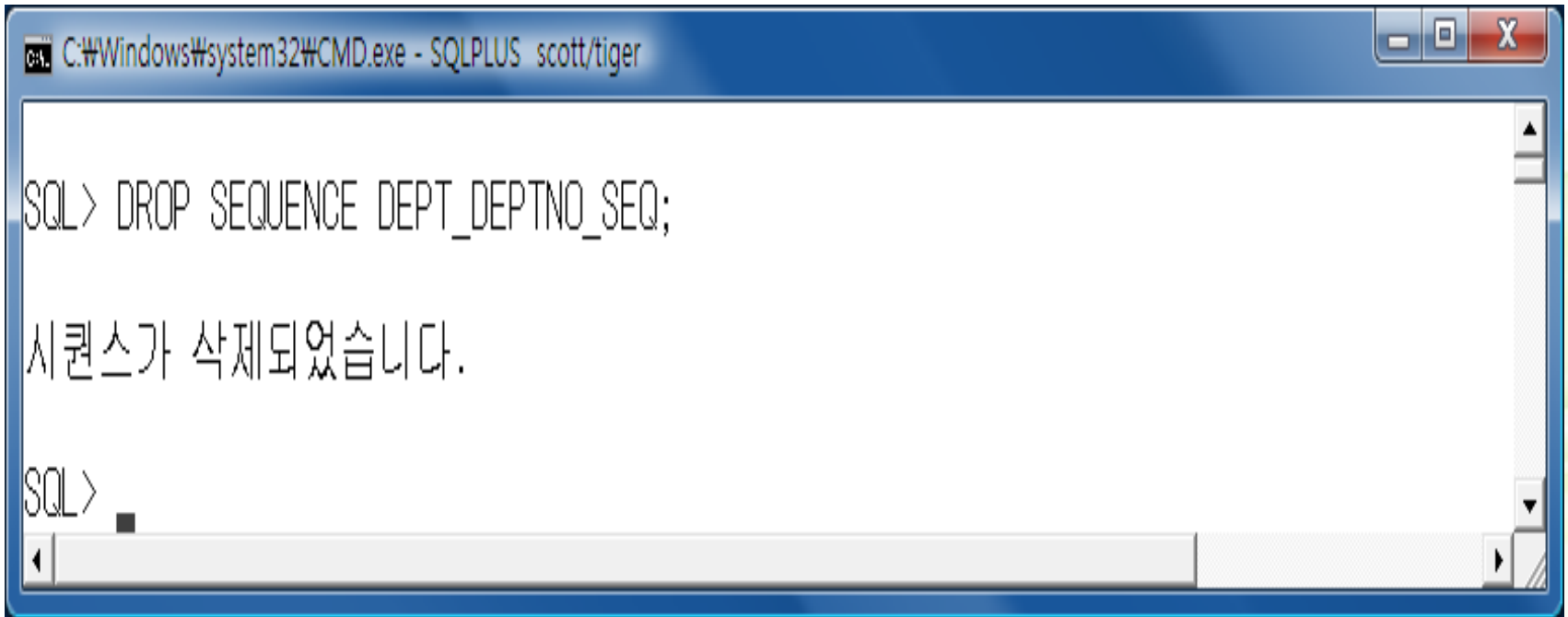
3. 사원 번호를 저장하는 EMPNO 컬럼은 기본 키로 설정하였으므로 중복된 값을 가질 수 없습니다. 다음은 생성한 EMP_SEQ 시퀀스로부터 사원번호를 자동으로 할당받아 데이터를 추가하는 문장입니다.

```
INSERT INTO EMP01  
VALUES(EMP_SEQ.NEXTVAL, 'JULIA' , SYSDATE);
```

연 습 문 제

- DROP SEQUENCE문으로 시퀀스를 제거해 봅시다.

```
DROP SEQUENCE DEPT_DEPTNO_SEQ;
```



The screenshot shows a Windows Command Prompt window titled "C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger". The command prompt displays the following text:

```
SQL> DROP SEQUENCE DEPT_DEPTNO_SEQ;
```

Below the command, the Korean message "시퀀스가 삭제되었습니다." (Sequence deleted) is displayed. The prompt "SQL>" is visible at the bottom left of the window.

연 습 문 제

5. 부서 번호를 생성하는 시퀀스 객체를 생성하여 시퀀스 객체를 이용하여 부서 번호를 자동 생성하도록 해 봅시다.

이 문제를 풀기 위해서 다음과 같이 부서 테이블을 생성합니다.

```
CREATE TABLE DEPT_EXAMPLE(  
  DEPTNO NUMBER(4) PRIMARY KEY,  
  DNAME VARCHAR(15),  
  LOC VARCHAR(15)  
);
```

부서 번호를 저장하는 DEPTNO 컬럼은 기본 키로 설정하였으므로 중복된 값을 가질 수 없습니다.

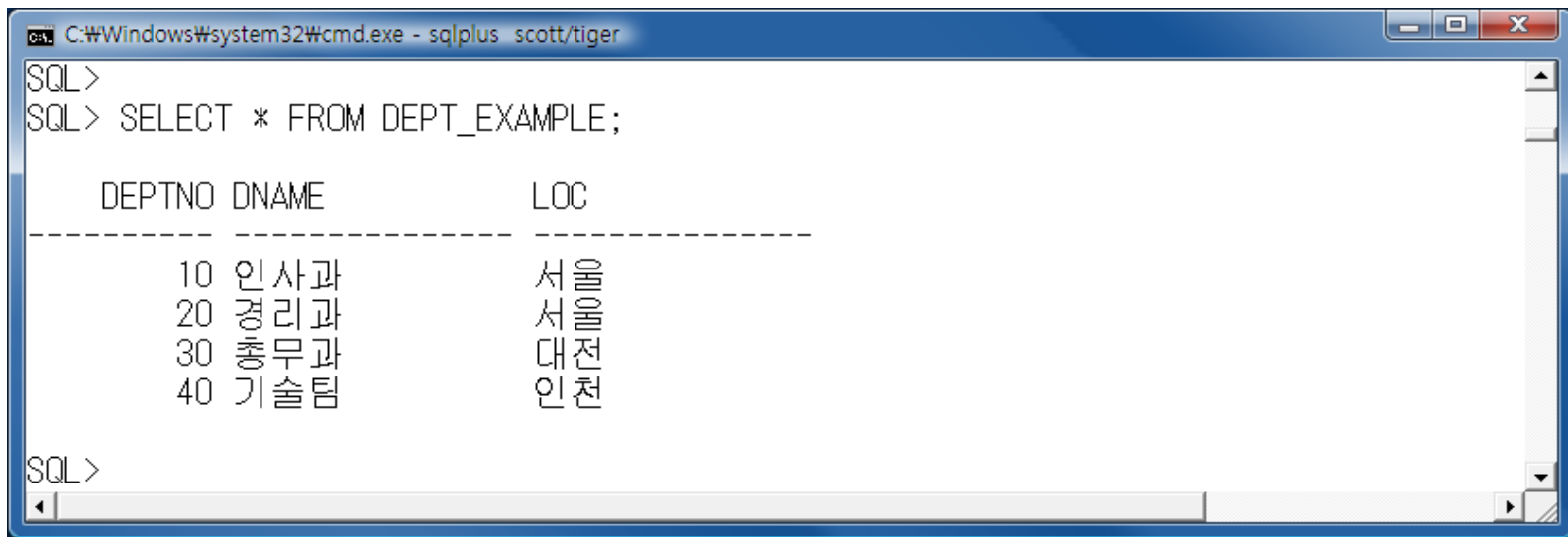
5.1 DEPTNO 컬럼에 유일한 값을 가질 수 있도록 시퀀스 객체 생성(시퀀스 이름 : DEPT_EXAMPLE_SEQ)해 봅시다.

5.2 새로운 로우를 추가할 때마다 시퀀스에 의해서 다음과 같이 부서 번호가 자동 부여되도록 해 봅시다.

연 습 문 제

다음은 시퀀스 객체 생성 후 로우를 추가한 결과를 확인한 결과 화면입니다.

```
SELECT * FROM DEPT_EXAMPLE;
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". The prompt is "SQL>". The user has entered the command "SELECT * FROM DEPT_EXAMPLE;". The output is a table with three columns: DEPTNO, DNAME, and LOC. The data is as follows:

DEPTNO	DNAME	LOC
10	인사과	서울
20	경리과	서울
30	총무과	대전
40	기술팀	인천

The prompt "SQL>" is visible at the bottom left of the window.

추가된 행을 살펴보면 시퀀스 객체가 발생시킨 일련번호가 부서 번호에 적용된 것을 확인할 수 있습니다.

시퀀스 객체의 초기 값을 10으로 하여 10씩 증가하도록 했기 때문에 증가치를 지정했기 때문에 부서 번호가 10, 20, 30, 40으로 지정된 것을 확인할 수 있습니다.

인덱스(indexes)

■ 인덱스(index)

- 인덱스는 책의 색인처럼 특정한 컬럼에 속한 값을 정렬하여 저장하는 것이다.
- 색인과 마찬가지로 물리적인 저장공간을 차지한다.
- 색인과 마찬가지로 찾는 자료의 양이 적을수록 유리하다. 대략 전체 행의 10%이하의 결과값을 가질 때 유리하다.
- 행의 입력, 수정, 삭제에 대하여 추가적인 시간을 필요로 한다.
- 인덱스의 후보 컬럼은 넓은 범위의 값을 가질수록 유리하다. 유일한 값을 가지는 컬럼이 남성/여성의 두 가지 컬럼값을 가지는 컬럼보다 좋은 후보가 될 가능성이 훨씬 높다.

1.4.2 인덱스 생성을 위한 지침

가) 많은 것이 항상 더 좋은 것은 아니다.

테이블의 많은 인덱스가 질의의 스피드 향상을 꼭 의미하는 것은 아닙니다. 인덱스를 가지고 있는 테이블에 대한 각 DML 작업은 인덱스도 갱신되어야 함을 의미합니다. 많은 인덱스가 테이블과 관련되어 있으면, ORACLE SERVER은 DML 후에 모든 인덱스를 갱신하기 위해 더 많은 노력이 필요하게 됩니다.

나) 언제 인덱스를 생성하는가?

- 열은 WHERE 절 또는 조인 조건에서 자주 사용됩니다.
- 열은 광범위한 값을 포함합니다.
- 열은 많은 수의 null 값을 포함합니다.
- 둘 또는 이상의 열은 WHERE 절 또는 조인 조건에서 자주 함께 사용됩니다.
- 테이블은 대형이고 대부분의 질의들은 행의 2~4%보다 적게 읽어 들일 것으로 예상됩니다.

다) 언제 인덱스를 생성해서는 안되는가

- 테이블이 작다.
- 열이 질의의 조건으로 자주 사용되지 않는다.
- 대부분의 질의들은 행의 2~4% 이상을 읽어 들일 것으로 예상된다.

테이블은 자주 갱신됩니다. 테이블에 하나 이상 인덱스를 가지고 있다면 테이블을 액세스하는 DML 문장은 인덱스의 유지 때문에 상대적으로 더 많은 시간이 걸리게 됩니다

인덱스(indexes)

■ 인덱스(index) 생성(create)

```
Create [unique] index index_name on  
Table_name (column_name1[, column_name2 ...])  
Tablespace tab_space;
```

- Unique : 인덱스 컬럼의 값이 유일한 경우에 기입한다.
- Index_name : 인덱스 명을 기입한다.
- Table_name : 인덱스를 생성할 테이블 명을 기입한다.
- Column_name : 인덱스를 생성할 테이블의 컬럼 명을 기입한다.
- Tab_space : 인덱스를 생성할 테이블스페이스를 지정한다.
기본값은 사용자 기본 테이블스페이스이다.
- 일반적으로는 성능상의 문제 때문에 인덱스는 테이블과 다른 테이블스페이스에 생성한다.

인덱스(indexes)

```
alter session set sql_trace=true ;
```

```
set autotrace { on | off }
```

```
SQL> set autotrace on
SQL> select customer_id, first_name, last_name
       2  from customers
       3  where last_name = 'Brown';
```

EXPLAIN plan for select * from emp where sal > 2000 and
ename='KK' order by sal desc;
select * from table(dbms_xplan.display);

No rows selected

Execution Plan

```
-----
0          SELECT STATEMENT Optimizer=ALL_ROWS
              (Cost=3 Card=1 Bytes=18)
1    0      TABLE ACCESS (FULL) OF 'CUSTOMERS'
              (TABLE) (Cost=3 Card=1 Bytes=18)
```

인덱스(indexes)

```
SQL> create index customers_last_name_idx on  
2      customers (last_name);
```

```
SQL> select customer_id, first_name, last_name  
2      from customers  
3      where last_name = 'Brown';
```

No rows selected

Execution Plan

```
-----  
0          SELECT STATEMENT Optimizer=ALL_ROWS  
          (Cost=2 Card=1 Bytes=18)  
1    0      TABLE ACCESS (BY INDEX ROWID) OF  
          'CUSTOMERS' (TABLE) (Cost=2 Card=1  
          Bytes=18)  
2    1      INDEX (RANGE SCAN) OF  
          'CUSTOMERS_LAST_NAME_IDX' (INDEX)  
          (Cost=1 Card=1)
```

인덱스(indexes)

- 유일(unique) 인덱스(index) 생성

```
SQL> create unique index customers_phone_idx on  
2      customers (phone);
```

- 복합(composite) 인덱스(index) 생성

```
SQL> create index employee_first_last_name_idx on  
2      employee(first_name, last_name);
```

인덱스(indexes)

- 함수 기반 (function-based) 인덱스(index) 생성
 - 인덱스(index)에 변형을 가하는 경우, 인덱스(index)를 사용할 수 없게 된다.

```
SQL> select customer_id, first_name, last_name  
2   from customers  
3   where upper(last_name) = 'BROWN';
```

No rows selected

Execution Plan

```
-----  
0      SELECT STATEMENT Optimizer=ALL_ROWS  
      (Cost=3 Card=1 Bytes=18)  
1    0      TABLE ACCESS (FULL) OF 'CUSTOMERS'  
      (TABLE) (Cost=3 Card=1 Bytes=18)
```


인덱스(indexes)

```
SQL> create index customers_last_name_func_idx on  
2      customers (upper(last_name)) ;
```

```
SQL> select customer_id, first_name, last_name  
2      from customers  
3      where upper(last_name) = 'BROWN';
```

No rows selected

Execution Plan

```
-----  
0          SELECT STATEMENT Optimizer=ALL_ROWS  
          (Cost=3 Card=1 Bytes=18)  
1    0      TABLE ACCESS (FULL) OF 'CUSTOMERS'  
          (TABLE) (Cost=3 Card=1 Bytes=18)
```

인덱스(indexes)

```
SQL> connect system/*****
```

```
SQL> alter system set query_rewrite_enabled = true;
```

```
SQL> show parameter query_rewrite_enabled
```

인덱스(indexes)

- 데이터 저장소(data dictionary)에서 인덱스(indexes) 정보 모으기

```
SQL> select index_name, table_name, uniqueness,  
          status  
2  from user_indexes  
3  where table_name in ('CUSTOMERS', 'EMPLOYEES');
```

INDEX_NAME	TABLE_NAM	UNIQUENES	STATU
-----	-----	-----	-----
CUSTOMERS_LAST_NAME_IDX	CUSTOMERS	NONUNIQUE	VALID
CUSTOMERS_PHONE_IDX	CUSTOMERS	UNIQUE	VALID
CUSTOMERS_PK	CUSTOMERS	UNIQUE	VALID
CUSTOMERS_LAST_NAME_FUNC_I	CUSTOMERS	NONUNIQUE	VALID
EMPLOYEES_FIRST_LAST_NAME_	EMPLOYEES	NONUNIQUE	VALID
EMPLOYEES_PK	EMPLOYEES	UNIQUE	VALID

인덱스(indexes)

```
SQL> select index_name, table_name, column_name,  
           position  
2   from user_ind_columns  
3   where table_name = 'CUSTOMERS';
```

INDEX_NAME	TABL	COLUMN_NAME	POS
-----	----	-----	---
CUSTOMERS_LAST_NAME_IDX	CUST	LAST_NAME	1
CUSTOMERS_PHONE_IDX	CUST	PHONE	1
CUSTOMERS_PK	CUST	CUSTOMER_ID	1
CUSTOMERS_LAST_NAME_FUNC_I	CUST	SYSNC00006\$	1
EMPLOYEES_FIRST_LAST_NAME_	EMPL	FIRST_NAME	1
EMPLOYEES_FIRST_LAST_NAME_	EMPL	LAST_NAME	2
EMPLOYEES_PK	EMPL	EMPLOYEE_ID	1

인덱스(indexes)

- 인덱스(indexes) 수정(modify)

```
SQL> alter index customers_phone_idx rename to  
      customers_phone_number_idx;
```

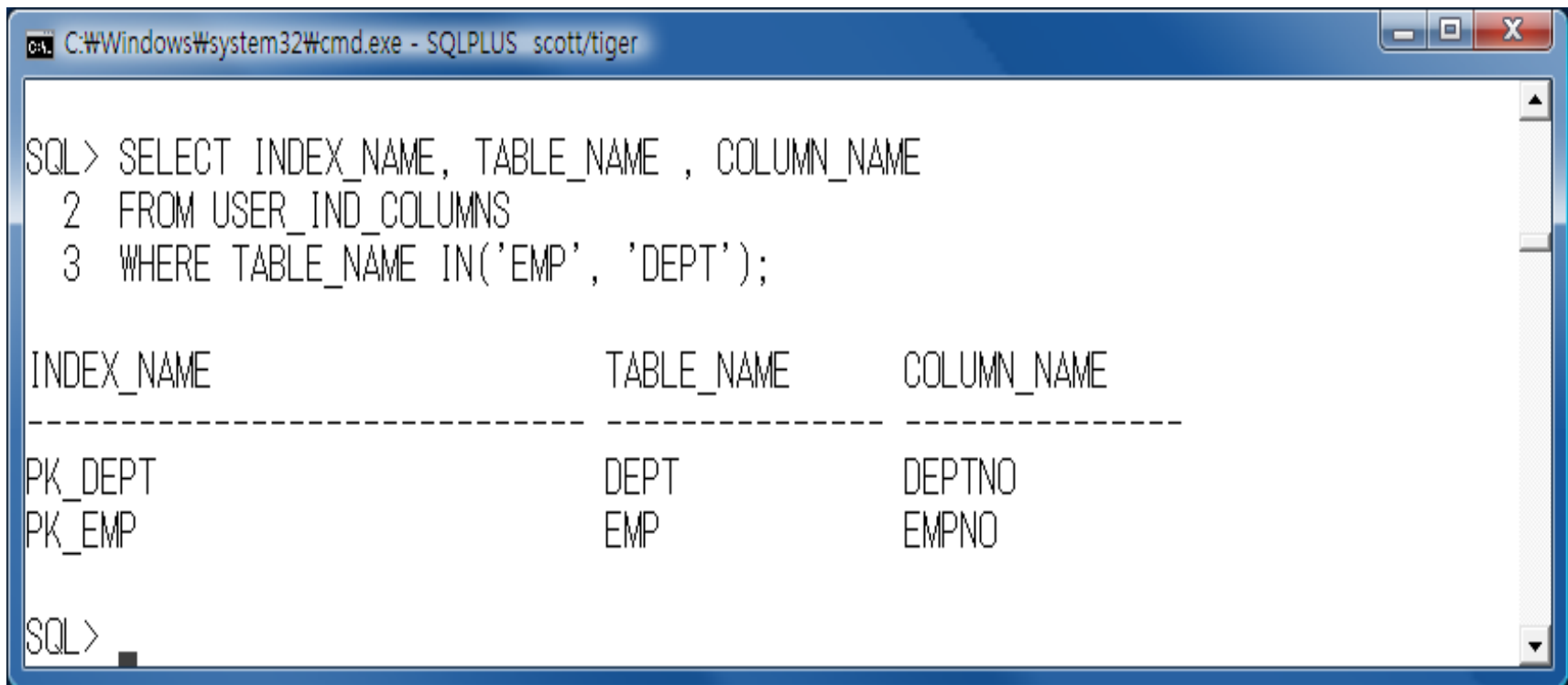
- 인덱스(indexes) 삭제(drop)

```
SQL> drop index customers_phone_number_idx;
```

실습하기

- USER_IND_COLUMNS 데이터 딕셔너리 뷰로 인덱스의 생성 여부를 확인해 봅시다.

```
SELECT INDEX_NAME, TABLE_NAME , COLUMN_NAME
FROM USER_IND_COLUMNS
WHERE TABLE_NAME IN('EMP', 'DEPT');
```



```
C:\Windows\system32\cmd.exe - SQLPLUS scott/tiger

SQL> SELECT INDEX_NAME, TABLE_NAME , COLUMN_NAME
  2  FROM USER_IND_COLUMNS
  3  WHERE TABLE_NAME IN('EMP', 'DEPT');

INDEX_NAME          TABLE_NAME          COLUMN_NAME
-----
PK_DEPT             DEPT                  DEPTNO
PK_EMP              EMP                   EMPNO

SQL>
```

실습하기

이번에는 테이블 EMP01의 컬럼 중에서 이름(ENAME)에 대해서 인덱스를 생성해봅시다.

```
CREATE INDEX IDX_EMP01_ENAME  
ON EMP01(ENAME);
```

EMP01 테이블의 IDX_EMP01_ENAME만 사용자가 인덱스를 생성한 것입니다. 이를 제거해 봅시다. 생성된 인덱스 객체를 제거하기 위해서는 DROP INDEX 문을 사용합니다.

```
DROP INDEX IDX_EMP01_ENAME;
```

실습하기

부서 번호와 부서명을 결합하여 결합 인덱스를 설정해 봅시다.

다음과 같이 부서 번호와 부서명을 결합하여 인덱스를 설정할 수 있는데 이를 결합 인덱스라고 합니다.

```
CREATE INDEX IDX_DEPT01_COM  
ON DEPT01(DEPTNO, DNAME);
```

데이터 디렉터리인 USER_IND_COLUMNS 테이블에서
IDX_DEPT01_COM 인덱스는 DEPTNO와 DNAME 두 개의 컬럼이
결합된 것임을 확인할 수 있습니다.

```
SELECT INDEX_NAME, COLUMN_NAME  
FROM USER_IND_COLUMNS  
WHERE TABLE_NAME = 'DEPT01';
```


함수 기반 인덱스

- 검색조건으로 WHERE SAL = 300이 아니라 WHERE SAL*12 = 3600와 같이 SELECT 문 WHERE 절에 산술 표현 또는 함수를 사용하는 경우가 있습니다.
- 이 경우 만약 SAL 컬럼에 인덱스가 걸려 있다면 인덱스를 타서 빠르리라 생각 할 수도 있지만 실상은 SAL 컬럼에 인덱스가 있어도 SAL*12는 인덱스를 타지 못합니다.
- 인덱스 걸린 컬럼이 수식으로 정의 되어 있거나 SUBSTR 등의 함수를 사용해서 변형이 일어난 경우는 인덱스를 타지 못하기 때문입니다.
- 이러한 수식으로 검색하는 경우가 많다면 아예 수식이나 함수를 적용하여 인덱스를 만들 수 있습니다. SAL*12로 인덱스를 만들어 놓으면 SAL*12가 검색 조건으로 사용될 시 해당 인덱스를 타게 됩니다.

실습하기

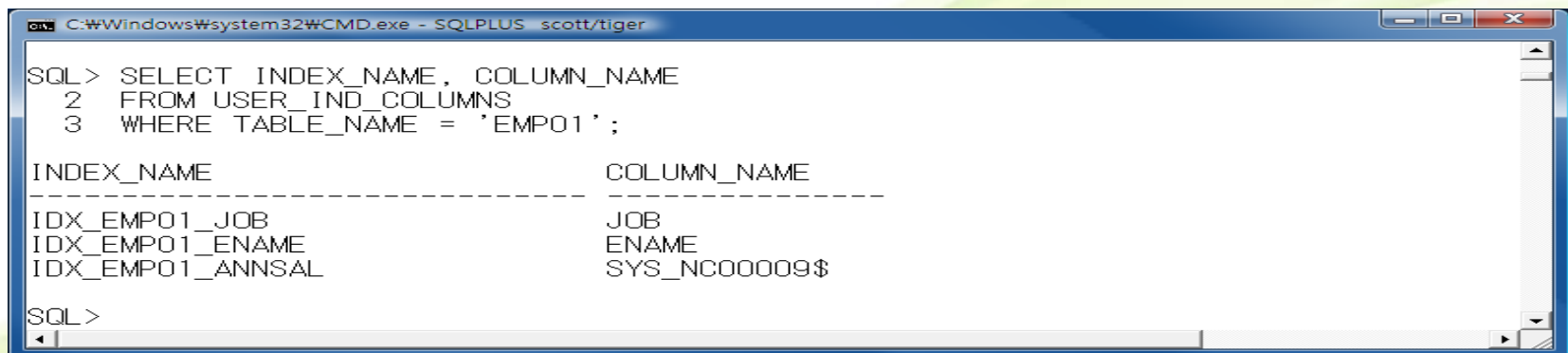
■ 사원 테이블에서 급여 컬럼에 저장된 데이터로 연봉을 인덱스로 지정하기 위한 산술 표현을 인덱스로 지정해 봅시다.

1. 함수 기반 인덱스를 생성해 봅시다.

```
CREATE INDEX IDX_EMP01_ANNSAL  
ON EMP01(SAL*12);
```

2. 다음은 데이터 디렉터리인 USER_IND_COLUMNS에 함수 기반 인덱스가 기록되어 있는 것을 확인하기 위한 쿼리문입니다.

```
SELECT INDEX_NAME, COLUMN_NAME  
FROM USER_IND_COLUMNS  
WHERE TABLE_NAME = 'EMP01'
```



C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger

```
SQL> SELECT INDEX_NAME, COLUMN_NAME  
2 FROM USER_IND_COLUMNS  
3 WHERE TABLE_NAME = 'EMP01';
```

INDEX_NAME	COLUMN_NAME
-----	-----
IDX_EMP01_JOB	JOB
IDX_EMP01_ENAME	ENAME
IDX_EMP01_ANNSAL	SYS_NC000009\$

```
SQL>
```

연 습 문 제

1. EMP 테이블에서 이름을 가지고 INDEX(emp_ename_idx)를 생성하여라
2. INDEX의 장단점을 기술하여라.
3. 테이블 생성시 자동적으로 생성되는 INDEX가 있다. 어떤 제약 조건을 기술하면 생성되는가?
4. INDEX 생성 지침을 설명하여라.
5. 여러 개의 COLUMN으로 인덱스를 사용하는 경우가 있다. 몇개의 COLUMN까지 가능한가.

동의어 개념과 종류

- 데이터베이스의 객체에 대한 소유권은 해당 객체를 생성한 사용자에게 있습니다. 따라서 다른 사용자가 객체에 접근하기 위해서는 소유자로부터 접근 권한을 부여 받아야 합니다. 또한 다른 사용자가 소유한 객체에 접근하기 위해서는 소유자의 이름을 객체 앞에 지정해야 합니다.

- 이렇게 객체를 조회할 때마다 일일이 객체의 소유자를 지정하는 것이 번거로울 경우 동의어를 정의하면 긴 이름대신 간단한 이름으로 접근할 수 있게 됩니다.

- 동의어는 개별 사용자를 대상으로 하는 비공개 동의어와 전체 사용자를 대상으로 한 공개 동의어가 있습니다.

- 동의어는 개별 사용자를 대상으로 하는 비공개 동의어와 전체 사용자를 대상으로 한 공개 동의어가 있습니다.

- 비공개 동의어

객체에 대한 접근 권한을 부여받은 사용자가 정의한 동의어로 해당 사용자만 사용할 수 있습니다.

- 공개 동의어

권한을 주는 사용자가 정의한 동의어로 누구나 사용할 수 있습니다. 공개 동의어는 DBA 권한을 가진 사용자만이 생성할 수 있습니다. SYNONYM 앞에 PUBLIC를 붙여서 정의합니다.

동의어 생성

- DUAL은 원래 SYS가 소유하는 테이블 명이므로 다른 사용자가 DUAL 테이블에 접근하려면 SYS.DUAL로 표현해야 하는 것이 원칙입니다.
- 그럼에도 불구하고 지금까지 모든 사용자가 SYS.을 생략하고 DUAL이라고 간단하게 사용하였습니다.
- 이럴 수 있었던 이유는 공개 동의어로 지정되어있기 때문입니다.
- 동의어를 정의하기 위한 CREATE SYNONYM 명령어의 기본 형식은 다음과 같습니다.

```
CREATE [PUBLIC] SYNONYM synonym_name  
FOR user_name.object_name;
```

- *synonym_name*은 *user_name.object_name*에 대한 별칭입니다.
- *user_name*은 객체를 소유한 오라클 사용자이고 *object_name*는 동의어를 만들려는 데이터베이스 객체 이름입니다.

동의어 생성

SALGRADE 동의어로 GUBUN를 생성하고 동의어로 조회하여라.

```
SQL> CREATE SYNONYM gubun
```

```
FOR salgrade;
```

```
SQL> SELECT *
```

```
FROM gubun;
```

GRADE	LOSAL	HISAL
-------	-------	-------

1	700	1200
---	-----	------

2	1201	1400
---	------	------

.....

동의어 삭제

동의어를 제거하기 위해 DROP SYNONYM문장을 사용합니다.

Syntax

```
DROP [PUBLIC] SYNONYM synonym_name;
```

앞에서 생성한 동의어를 삭제하여라.

```
SQL> DROP SYNONYM gubun;
```

Synonym dropped.

1. EMP 테이블을 EMPLOYEE라는 동의어를 생성하여라.