



VIEW

VIEW

❖ Inline VIEW

- ✓ WHERE 절의 Sub Query는 행을 리턴하는데 하나일 수도 있고 여러 개일 수도 있지만 이에 비해 FROM 절에 오는 Sub Query는 값이 아닌 테이블을 리턴하는데 FROM 절은 조회 대상 테이블을 명시하는 문장이니 FROM 다음의 Sub Query는 테이블과 자격이 같음
- ✓ FROM 절의 Sub Query를 특별히 Inline VIEW 라고 부르는데 뷰는 테이블의 정보를 가지는 DB 오브젝트
- ✓ 인라인이 앞에 붙은 이유는 CREATE로 만들어 영구적으로 저장하는 것이 아니라 서브쿼리에서 잠시 만들어 쓰고 버리는 임시적인 뷰 라는 의미임
- ✓ Sub Query가 FROM 절에 오면 SELECT 문 끼리 중첩됨
- ✓ Inline VIEW의 SELECT가 리턴하는 결과 셋은 하나의 테이블인데 SELECT * FROM tCity는 도시 목록이며 이 구문이 FROM 절에 올 수 있음
- ✓ SELECT 한 걸 또 SELECT하는 구조
- ✓ 간단한 Inline VIEW

SELECT * FROM (SELECT * FROM tCity) A;

- FROM 절에 있는 안쪽의 SELECT 문이 리턴하는 도시 목록은 하나의 테이블이니 결과 셋으로부터 또 SELECT 명령을 실행할 수 있는데 이후 Inline VIEW의 필드를 참조하기 위해 당장 쓰지 않더라도 별도의 이름을 붙여주는 것이 좋음

VIEW

❖ Inline VIEW

- ✓ 과장 또는 부장 인 직원 중 성취도가 70점 이상인 직원 조회

```
SELECT * FROM (SELECT * FROM tStaff WHERE grade = '과장' OR grade = '부장') A  
WHERE A.score >= 70;
```



VIEW

❖ VIEW

- ✓ 물리적인 테이블을 근거한 논리적인 가상 테이블
- ✓ 가상이란 단어는 실질적으로 데이터를 저장하고 있지 않기 때문에 붙인 것이고 테이블이란 단어는 실질적으로 데이터를 저장하고 있지 않더라도 사용자는 마치 테이블을 사용하는 것과 동일하게 뷰를 사용할 수 있기 때문에 붙인 것
- ✓ 기본 테이블 이나 다른 뷰 로부터 파생된 객체로서 기본 테이블에 대한 하나의 SELECT 구문
- ✓ 사용자에게 주어진 뷰를 통해서 기본 테이블을 제한적으로 사용하도록 함
- ✓ 뷰를 생성하기 위해서는 실질적으로 데이터를 저장하고 있는 물리적인 테이블이나 다른 뷰가 존재해야 함



VIEW

❖ 실습을 위한 테이블을 생성

1. DEPT_COPY를 DEPT 테이블의 복사본으로 생성

```
CREATE TABLE DEPT_COPY  
AS  
SELECT * FROM DEPT;
```

2. EMP 테이블의 복사본으로 EMP_COPY를 생성

```
CREATE TABLE EMP_COPY  
AS  
SELECT * FROM EMP;
```



VIEW

❖ 뷰를 생성하기 위한 기본 형식

```
CREATE [OR REPLACE] VIEW view_name
```

```
[(alias, alias, alias, ...)]
```

```
AS subquery
```

```
[WITH CHECK OPTION]
```

```
[WITH READ ONLY];
```

- ✓ 테이블을 생성하기 위해서는 CREATE TABLE 로 시작하지만 뷰를 생성하기 위해서는 CREATE VIEW로 시작
- ✓ AS 다음은 Sub Query 문과 유사
- ✓ Sub Query 에는 SELECT 문을 기술



VIEW

❖ 뷰를 생성하기 위한 기본 형식

✓ CREATE OR REPLACE VIEW

- CREATE VIEW를 통해 만들어진 뷰의 구조를 바꾸려면 뷰를 삭제하고 다시 만들어야 되는 반면 CREATE OR REPLACE VIEW는 새로운 뷰를 만들 수 있을 뿐만 아니라 기존에 뷰가 존재하더라도 삭제하지 않고 새로운 구조의 뷰로 변경(REPLACE)

✓ WITH CHECK OPTION

- WITH CHECK OPTION을 사용하면 해당 뷰를 통해서 볼 수 있는 범위 내에서만 UPDATE 또는 INSERT 가능

✓ WITH READ ONLY

- WITH READ ONLY를 사용하면 해당 뷰를 통해서만 SELECT만 가능하며 INSERT/UPDATE/DELETE를 할 수 없음
- 생략하면 뷰를 사용하여 추가, 수정, 삭제(INSERT/UPDATE/DELETE)가 모두 가능



VIEW

❖ VIEW 생성

- ✓ 30번 부서에 소속된 직원들의 사번과 이름과 부서번호를 자주 검색한다고 한다면 같은 SELECT 문을 여러 번 입력해야 함

```
SELECT EMPNO, ENAME, DEPTNO  
FROM EMP_COPY  
WHERE DEPTNO=30;
```

- 동일한 결과를 출력하기 위해서 매번 SELECT 문을 입력하기란 번거로운 일
- 뷰는 이와 같이 번거로운 SELECT 문을 매번 입력하는 대신 보다 쉽게 원하는 결과를 얻고자 하는 바람에서 출발한 개념



VIEW

❖ VIEW 생성

- ✓ 자주 사용되는 30번 부서에 소속된 직원들의 사번과 이름과 부서번호를 출력하기 위한 SELECT문을 하나의 뷰로 정의

```
CREATE VIEW EMP_VIEW30
```

```
AS
```

```
SELECT EMPNO, ENAME, DEPTNO
```

```
FROM EMP_COPY
```

```
WHERE DEPTNO=30;
```

- ✓ 뷰는 테이블에 접근(SELECT)한 것과 동일한 방법으로 결과를 얻을 수 있음

```
SELECT * FROM EMP_VIEW30;
```

- ✓ 뷰 구조 확인

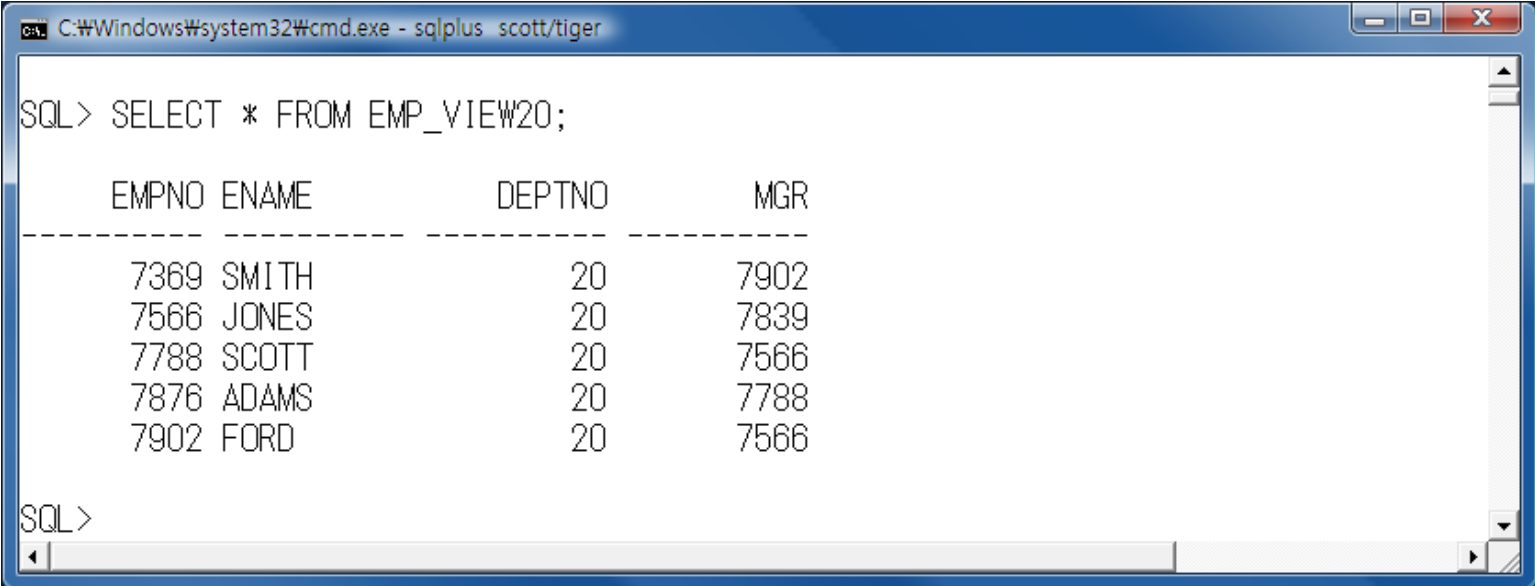
```
DESC EMP_VIEW30;
```



VIEW

❖ 연습문제

- ✓ 기본 테이블은 EMP_COPY를 이용하고 deptno가 20번인 부서에 소속된 사원들의 사번(empno)과 이름(ename)과 부서번호(deptno)와 상관의 사번(mgr)을 출력하기 위한 SELECT문을 EMP_VIEW20 이란 이름의 뷰로 정의



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

SQL> SELECT * FROM EMP_VIEW20;

  EMPNO ENAME      DEPTNO    MGR
-----
    7369 SMITH          20     7902
    7566 JONES           20     7839
    7788 SCOTT           20     7566
    7876 ADAMS           20     7788
    7902 FORD           20     7566

SQL>
```

VIEW

❖ VIEW 의 내부 구조

- ✓ EMP_VIEW30라는 뷰는 데이터를 물리적으로 저장하고 있지 않으며 CREATE VIEW 명령어로 뷰를 정의할 때 AS 절 다음에 기술한 쿼리 문장 자체를 저장하고 있음



VIEW

❖ 뷰를 사용하는 이유

- ✓ 복잡하고 긴 쿼리문을 뷰로 정의하면 접근을 단순화시킬 수 있음
- ✓ 보안에 유리
 - 사용자마다 특정 객체만 조회할 수 있도록 권한을 부여할 수 있기에 동일한 테이블을 접근하는 사용자마다 서로 다르게 보도록 여러 개의 뷰를 정의해 놓고 특정 사용자만이 해당 뷰에 접근할 수 있도록 하면 보안에 유리
- ✓ 엔진은 뷰를 참조할 때마다 뷰의 쿼리문을 매번 다시 실행하여 원본 테이블을 읽기 때문에 원본 테이블이 바뀌면 뷰의 내용도 바뀌고 원본을 삭제하면 뷰는 더 이상 동작하지 않으며 뷰는 항상 원본 테이블을 실시간으로 읽어 동기화 상태를 유지
- ✓ 실제 데이터를 보유하지 않는 가짜이지만 테이블과 같은 자격을 가지며 테이블이 올 수 있는 모든 곳에 올 수 있으며 원본으로부터 파생시킨 새로운 모양의 테이블이며 테이블에 적용되는 모든 문법을 사용할 수 있음



VIEW

❖ VIEW 종류

- ✓ 뷰는 뷰를 정의하기 위해서 사용되는 기본 테이블의 수에 따라 단순 뷰(Simple VIEW)와 복합 뷰(Complex VIEW)로 나뉨

단순 뷰	복합 뷰
하나의 테이블로 생성	여러 개의 테이블로 생성
그룹 함수의 사용이 불가능	그룹 함수의 사용이 가능
DISTINCT 사용이 불가능	DISTINCT 사용이 가능
DML 사용 가능	DML 사용이 제한적(일반적으로 불가능)



VIEW

❖ 단순 뷰에 대해서 DML(INSERT/UPDATE/DELETE) 문을 사용할 수 있음을 확인

✓ EMP_VIEW30 뷰에 데이터를 추가

```
INSERT INTO EMP_VIEW30  
VALUES(8000, 'ANGEL', 30);
```

```
SELECT * FROM EMP_VIEW30;
```

✓ 단순 뷰를 대상으로 실행한 DML 명령문의 처리 결과는 뷰를 정의할 때 사용한 기본 테이블에 적용

```
SELECT * FROM EMP_COPY;
```



VIEW

❖ 다른 이름 사용

- ✓ 기본 테이블(EMP_COPY)의 컬럼 명을 한글화 하여 컬럼 명이 사원번호, 사원명, 급여, 부서번호로 구성

```
CREATE OR REPLACE
```

```
VIEW EMP_VIEW(사원번호, 사원명, 급여, 부서번호)
```

```
AS
```

```
SELECT EMPNO, ENAME, SAL, DEPTNO
```

```
FROM EMP_COPY;
```

- ✓ EMP_VIEW30 뷰에 데이터 확인

```
SELECT *
```

```
FROM EMP_VIEW
```

```
WHERE 부서번호=30;
```

- ✓ 뷰는 컬럼에 별칭을 주게 되면 기본 테이블의 컬럼 명을 더 이상 사용하지 못하므로 30번 부서를 검색하기 위해서는 뷰를 생성할 때 준 컬럼 별칭(부서번호)을 이용

VIEW

❖ 계산식 사용

- ✓ 그룹 함수 SUM과 AVG를 사용해서 각 부서별 급여 총액과 평균을 구하는 뷰를 작성 하기 위해서 SELECT 절 다음에 SUM이란 그룹 함수를 사용하면 결과를 뷰의 특정 컬럼처럼 사용하는 것이므로 물리적인 컬럼이 존재하지 않는 가상 컬럼이기에 뷰를 생성할 때 가상 컬럼을 사용하려면 사용자가 반드시 이름을 따로 설정해야 함

- 부서별 급여 총액과 평균을 구하기 위한 뷰를 생성

```
CREATE VIEW VIEW_SAL
```

```
AS
```

```
SELECT DEPTNO, SUM(SAL) AS "SalSum", AVG(SAL) AS "SalAvg"
```

```
FROM EMP_COPY
```

```
GROUP BY DEPTNO;
```



VIEW

❖ 뷰를 이용한 DML 작업

- ✓ 단순 뷰는 DML 명령어를 사용하여 조작이 가능
- ✓ 몇 가지의 경우에는 조작이 불가능
 - 뷰 정의에 포함되지 않은 컬럼 중에 기본 테이블의 컬럼이 NOT NULL 제약 조건이 지정되어 있는 경우 INSERT 문이 사용 불가능한데 뷰에 대한 INSERT 문은 기본 테이블에 NULL 값을 입력하는 형태가 되기 때문
 - SAL*12와 같이 산술 표현식으로 정의된 가상 컬럼이 뷰에 정의되면 INSERT나 UPDATE가 불가능
 - DISTINCT를 포함한 경우에도 DML 명령을 사용할 수 없음
 - 그룹 함수나 GROUP BY 절을 포함한 경우에도 DML 명령을 사용할 수 없음
 - 복합 뷰도 DML 작업에 제한이 있음



VIEW

❖ 복합 뷰

- ✓ 뷰를 사용하는 이유 중의 하나가 복잡하고 자주 사용하는 질의를 보다 쉽고 간단하게 사용하기 위해서 인데 사원 테이블과 부서 테이블을 자주 조인하는 경우에 사원 테이블과 부서 테이블을 조인하기 위해서는 다음과 같은 SELECT 문을 매번 작성해야 함

```
SELECT E.EMPNO, E.ENAME, E.SAL, E.DEPTNO, D.DNAME, D.LOC  
FROM EMP E, DEPT D  
WHERE E.DEPTNO = D.DEPTNO  
ORDER BY EMPNO DESC;
```

- ✓ 조인문에 CREATE VIEW EMP_VIEW_DEPT AS 만 추가해서 뷰로 작성해 놓으면 SELECT * FROM EMP_VIEW_DEPT; 와 같이 간단하게 질의 결과를 얻을 수 있음



VIEW

❖ 복합 뷰

- ✓ 사원 테이블과 부서 테이블을 조인하기 위해서 복합 뷰를 생성

1. 다음은 사번, 이름, 급여, 부서번호, 부서명, 지역명을 출력하기 위한 복합 뷰

```
CREATE VIEW EMP_VIEW_DEPT
```

```
AS
```

```
SELECT E.EMPNO, E.ENAME, E.SAL, E.DEPTNO, D.DNAME, D.LOC
```

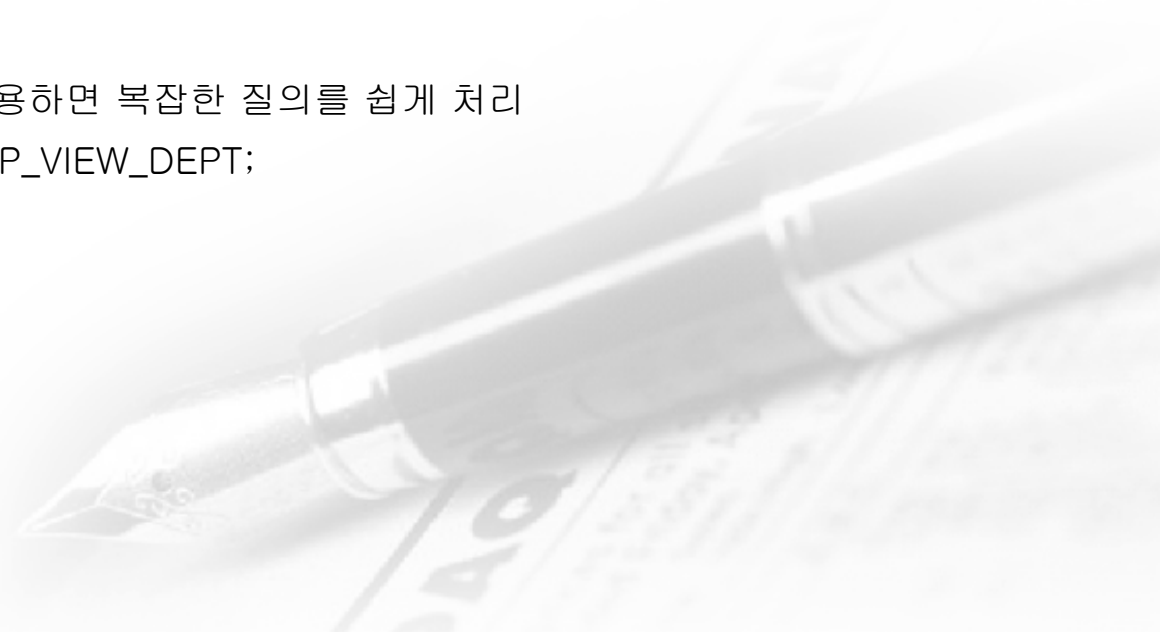
```
FROM EMP E, DEPT D
```

```
WHERE E.DEPTNO = D.DEPTNO
```

```
ORDER BY EMPNO DESC;
```

2. 뷰를 생성한 후, 이를 활용하면 복잡한 질의를 쉽게 처리

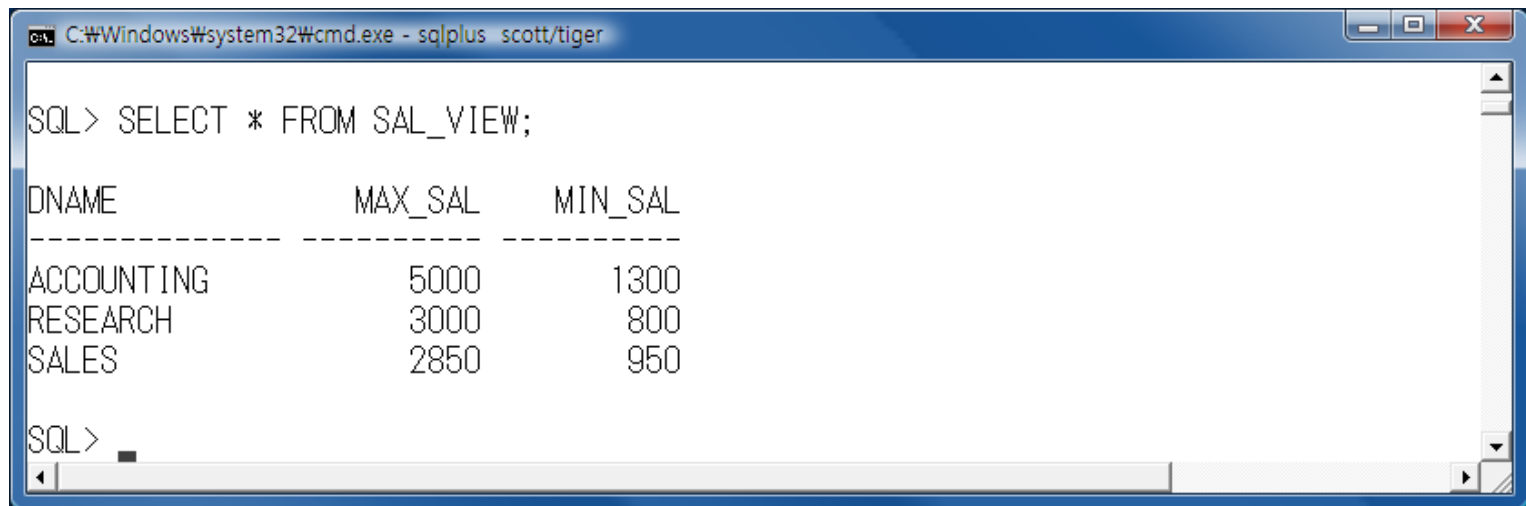
```
SELECT * FROM EMP_VIEW_DEPT;
```



VIEW

❖ 연습문제

- ✓ EMP 테이블에서 부서별(deptno) 최대 급여(sal)와 최소 급여를 출력하는 뷰를 SAL_VIEW란 이름으로 작성



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe - sqlplus scott/tiger". Inside the window, the SQL prompt "SQL>" is followed by the command "SELECT * FROM SAL_VIEW;". The output is a table with three columns: DNAME, MAX_SAL, and MIN_SAL. The data rows are: ACCOUNTING (5000, 1300), RESEARCH (3000, 800), and SALES (2850, 950). The prompt "SQL>" is visible at the bottom left of the window.

```
SQL> SELECT * FROM SAL_VIEW;
```

DNAME	MAX_SAL	MIN_SAL
ACCOUNTING	5000	1300
RESEARCH	3000	800
SALES	2850	950

```
SQL>
```

VIEW

❖ 뷰 삭제

- ✓ 뷰는 실체가 없는 가상 테이블이기 때문에 뷰를 삭제한다는 것은 데이터 디렉터리에서 저장되어 있는 뷰의 정의를 삭제하는 것
- ✓ 뷰를 삭제해도 뷰를 정의한 기본 테이블의 구조나 데이터에는 전혀 영향을 주지 않음
- ✓ VIEW_SAL을 삭제
DROP VIEW VIEW_SAL;



VIEW

❖ 뷰 생성

```
CREATE [OR REPLACE] VIEW view_name  
[(alias, alias, alias, ...)]  
AS subquery  
[WITH CHECK OPTION]  
[WITH READ ONLY];
```



VIEW

❖ 뷰 생성 옵션

✓ CREATE OR REPLACE VIEW

- 존재하지 않은 뷰이면 새로운 뷰를 생성하고 기존에 존재하는 뷰이면 그 내용을 변경
- 이전에 작성한 EMP_VIEW30 뷰는 EMPNO, ENAME, SAL, DEPTNO 4 개의 컬럼을 출력하는 형태였는데 커미션 컬럼을 추가로 출력할 수 있도록 하기 위해서 뷰의 구조를 변경

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



VIEW

❖ 뷰 생성 옵션

✓ WITH CHECK OPTION

- 30 번 부서 소속 직원들의 정보만으로 구성된 뷰

```
CREATE OR REPLACE VIEW EMP_VIEW30
```

```
AS
```

```
SELECT EMPNO, ENAME, DEPTNO, SAL
```

```
FROM EMP_COPY
```

```
WHERE DEPTNO=30;
```

```
SELECT * FROM EMP_VIEW30;
```

- 뷰는 테이블처럼 SELECT문으로 조회할 수 있음은 물론이고 DML 문으로 내용을 조작할 수 있는데 UPDATE 문으로 30번 부서에 소속된 직원 중에 급여가 1200 이상인 직원은 20번 부서로 변경

```
UPDATE EMP_VIEW30
```

```
SET DEPTNO=20
```

```
WHERE SAL >= 1200;
```


VIEW

❖ 뷰 생성 옵션

✓ WITH CHECK OPTION

- 뷰의 조건내에서만 삽입, 삭제, 갱신이 가능하도록 제한

```
CREATE OR REPLACE VIEW VIEW_CHK30
```

```
AS
```

```
SELECT EMPNO, ENAME, DEPTNO, SAL
```

```
FROM EMP_COPY
```

```
WHERE DEPTNO=30
```

```
WITH CHECK OPTION;
```



VIEW

❖ 뷰 생성 옵션

✓ WITH CHECK OPTION

- SAL 이 1200 이상인 사원의 부서를 20번 부서로 변경

```
UPDATE VIEW_CHK30
```

```
SET DEPTNO=20
```

```
WHERE SAL >= 1200;
```

```
SELECT *
```

```
FROM EMP_COPY;
```



VIEW

❖ 연습문제

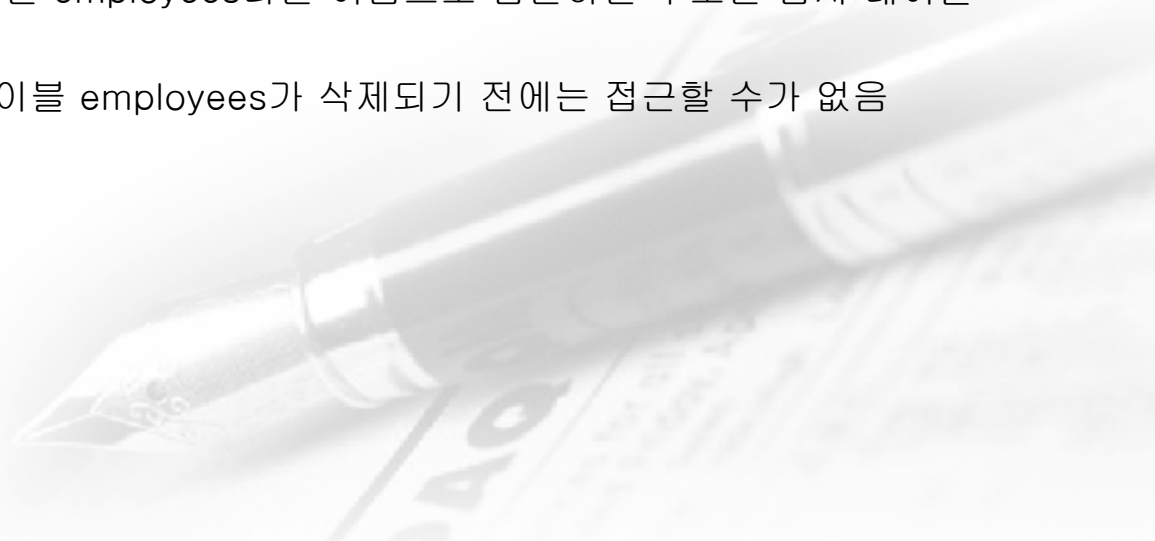
- ✓ EMP 테이블에서 사원 번호(empno), 이름(ename), 업무(job), 부서번호(DEPTNO)를 포함하는 EMP_VIEW라는 이름의 VIEW를 생성
- ✓ 앞에서 생성한 VIEW를 이용하여 DEPTNO가 10번 부서의 자료만 조회



임시 테이블

❖ 임시 테이블

- CREATE TEMPORARY TABLE [IF NOT EXISTS] 테이블이름 (열정의...)
- ✓ 구문 중에서 TABLE 위치에 TEMPORARY TABLE이라고 써주는 것 외에는 테이블과 정의하는 것이 동일
- ✓ 임시 테이블은 정의하는 구문만 약간 다를 뿐 나머지 사용법 등은 일반 테이블
- ✓ 임시 테이블은 세션 내에서만 존재하며 세션이 닫히면 자동으로 삭제되며 임시 테이블은 생성한 클라이언트에서만 접근이 가능하며 다른 클라이언트는 접근할 수 없음
- ✓ 임시 테이블은 데이터베이스 내의 다른 테이블과 이름을 동일하게 만들 수 있는데 그러면 기존의 테이블은 임시 테이블이 있는 동안에 접근이 불가능하고 무조건 임시 테이블로 접근할 수 있음
- ✓ employees DB 안에 employees라는 테이블이 기존에 있지만 임시 테이블도 employees 이름으로 생성할 수 있는데 그러면 employees라는 이름으로 접근하면 무조건 임시 테이블 employees 로 접근됨
- ✓ 기존의 employees는 임시 테이블 employees가 삭제되기 전에는 접근할 수가 없음



임시 테이블

❖ CTE

- ✓ 중간 과정의 결과셋을 여러 번 사용할 때 임시 테이블이나 뷰를 사용하는데 둘 다 임시적이지만 DB에 실제 저장되는 객체여서 이미 존재하면 먼저 삭제해야 하며 사용 후 정리해야 한다는 면에서 번거롭고 객체를 생성하고 저장 공간을 차지하기 때문에 속도도 그다지 빠르지는 않음
- ✓ CTE(Common Table Expressions)는 쿼리 실행 중에 메모리에 존재하는 테이블
- ✓ 쿼리 내부에서 임시 테이블을 정의하는 일종의 매크로 기능이며 예비 동작 없이 쿼리문 내부에서 모든 것을 일괄 처리할 수 있어 간편하며 ANSI SQL99의 표준 기능이며 모든 DBMS가 지원하여 호환성이 높음
- ✓ 형식

WITH 테이블명(필드목록) AS (쿼리문)

CTE 사용

- WITH 구문으로 쿼리문에 대해 이름을 붙여 생성하며 필드 목록 생략 시 내부 쿼리의 필드명을 사용
- 뷰 정의문과 유사하되 객체를 생성하지 않고 이름만 부여
- 이어지는 명령과 구분하기 위해 쿼리문을 괄호로 감싼다는 점도 다름
- 이렇게 정의한 이름으로 이어지는 쿼리에서 결과셋을 사용

임시 테이블

❖ CTE

✓ CTE 를 이용한 해결

WITH TEMP AS

(SELECT NAME, SALARY, SCORE FROM tStaff WHERE DEPART = '영업부' AND
GENDER = '남')

SELECT * FROM TEMP WHERE SALARY >= (SELECT AVG(SALARY) FROM TEMP);

- 영업부 남직원 목록을 TEMP 로 정의
- 이후 쿼리에서는 TEMP를 마치 테이블처럼 사용하면 됨
- TEMP에 대해 단순 조회는 물론이고 서브 쿼리나 다른 테이블과의 조인 등 테이블로 할 수 있는 모든 동작을 다 할 수 있음
- 평균 월급을 조사할 때나 평균이 넘는 직원을 조사할 때나 TEMP 라는 이름으로 참조하면 됨



임시 테이블

❖ CTE

- ✓ CTE는 디스크에 기록되는 DB 객체가 아니며 쿼리문 실행 중에 메모리상에서 잠시만 존재하기 때문에 CTE 정의문 과 사용문을 한 번에 실행하며 전체가 하나의 문장
- ✓ 앞 부분만 실행하면 쓰지도 않을 CTE를 정의하는 것이라서 에러이며 뒷부분만 실행하면 TEMP가 없어 에러
- ✓ 트랜잭션 내부의 지역 객체여서 명칭 충돌 걱정이 없으며 tCity나 tMember로 이름을 붙여도 상관없음
- ✓ CTE를 정의할 때 테이블명 뒤에 필드 목록을 밝히면 원래 쿼리와 다른 필드명을 쓸 수 있는데 필드 개수는 내부 쿼리와 일치해야 함
- ✓ 필드명을 한글로 붙이면 가독성이 향상됨
- ✓ 별명은 옵션일 뿐이지만 SUM (salary), COUNT (*) 같은 계산 필드는 이름을 붙여야 외부에서 사용 할 수 있음

WITH TEMP(이름, 월급, 성취도) AS

(SELECT NAME, SALARY, SCORE FROM tStaff WHERE DEPART = '영업부' AND
GENDER = '남')

SELECT * FROM TEMP WHERE 월급 >= (SELECT AVG(월급) FROM TEMP);

임시 테이블

❖ CTE

- ✓ 두 개 이상의 CTE 테이블을 콤마로 구분하여 한꺼번에 정의할 수도 있음
- ✓ 영업부 남직원과 여직원을 각각의 테이블로 정의하고 남직원 평균 월급보다 많이 받는 여직원 목록을 조사

WITH MAN AS

(SELECT NAME, SALARY, SCORE FROM tStaff WHERE DEPART = '영업부' AND
GENDER = '남'),

WOMAN AS

(SELECT NAME, SALARY, SCORE FROM tStaff WHERE DEPART = '영업부' AND
GENDER = '여')

SELECT * FROM WOMAN WHERE SALARY >= (SELECT avg(SALARY) FROM MAN);



임시 테이블

❖ CTE

✓ CTE를 활용하는 CTE

WITH MAN AS

(SELECT NAME, SALARY, SCORE FROM tStaff WHERE DEPART = '영업부' AND
GENDER = '남'),

GOODMAN AS

(SELECT NAME, SALARY, SCORE FROM MAN WHERE SCORE > 70)

SELECT *

FROM GOODMAN;



임시 테이블

❖ CTE

- ✓ 복잡한 구문을 CTE로 생성

```
WITH SHOPPING AS
```

```
(SELECT M.MEMBER, M.ADDR, O.ITEM, O.NUM, O.ORDERDATE FROM tMember M  
    INNER JOIN tOrder O ON M.MEMBER = O.MEMBER)
```

```
SELECT *
```

```
FROM SHOPPING
```

```
WHERE NUM >= (SELECT AVG(NUM) FROM SHOPPING);
```

- ✓ 위의 내용을 뷰 이용

```
CREATE VIEW VTEMP AS
```

```
(SELECT M.MEMBER, M.ADDR, O.ITEM, O.NUM, O.ORDERDATE FROM tMember M  
    INNER JOIN tOrder O ON M.MEMBER = O.MEMBER);
```

```
SELECT *
```

```
FROM VTEMP
```

```
WHERE NUM >= (SELECT AVG(NUM) FROM VTEMP);
```

- ✓ VIEW 와 CTE는 기능은 같지만 트랜잭션 내에서만 잠시 생성한 후 버린다는 면에서 CTE가 더 간편하고 성능도 좋음