



## 21장 저장 프로시저 , 함수, 커서

이번 장에서는 자주 사용되는 쿼리문을 모듈화하기 위한 저장 프로시저와 함수를 생성하고 고치고 지우는 작업들을 학습하고 커서에 대한 개념과 커서를 사용하기 위한 방법을 살펴보도록 하겠습니다.

## 이 장에서 다룰 내용



- 1    **저장 프로시저**  
.....●
- 2    **저장 프로시저 조회하기**  
.....●
- 3    **저장 프로시저의 매개 변수**  
.....●
- 4    **IN, OUT, INOUT 매개 변수**  
.....●
- 5    **저장 함수 생성**  
.....●
- 6    **커서**  
.....●

## 01. 저장 프로시저



- ❖ 지금까지 실습한 예제는 한번 실행하면 결과값을 돌려주고 끝나는 예제였습니다.
- ❖ 경우에 따라서는 우리가 만든 PL/SQL을 저장해 놓고 필요한 경우 호출하여 사용할 수 있으면 할 때가 있습니다. 오라클은 사용자가 만든 PL/SQL 문을 데이터베이스에 저장 할 수 있도록 저장프로시저라는 것을 제공합니다.
- ❖ 이렇게 저장 프로시저를 사용하면 복잡한 DML 문들 필요할 때마다 다시 입력할 필요 없이 간단하게 호출만 해서 복잡한 DML 문의 실행 결과를 얻을 수 있습니다.
- ❖ 저장 프로시저를 사용하면 성능도 향상되고, 호환성 문제도 해결됩니다.

# 01. 저장 프로시저

- ❖ 저장 프로시저를 생성하기 위한 CREATE PROCEDURE의 형식은 다음과 같습니다.



```
CREATE [OR REPLACE ] PROCEDURE prcedure_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
IS
  local_variable declaration
BEGIN
  statement1;
  statement2;
  . . .
END;
/
```

## 01. 저장 프로시저



- ❖ 저장 프로시저를 생성하려면 **CREATE PROCEDURE** 다음에 새롭게 생성하고자 하는 프로시저 이름을 기술합니다.
- ❖ 이렇게 해서 생성한 저장 프로시저는 여러 번 반복해서 호출해서 사용할 수 있다는 장점이 있습니다.
- ❖ 생성된 저장 프로시저를 제거하기 위해서는 **DROP PROCEDURE** 다음에 제거하고자 하는 프로시저 이름을 기술합니다.
- ❖ **OR REPLACE** 옵션은 이미 학습한 대로 이미 같은 이름으로 저장 프로시저를 생성할 경우 기존 프로시저는 삭제하고 지금 새롭게 기술한 내용으로 재 생성하도록 하는 옵션입니다.

## 01. 저장 프로시저



- ❖ 프로시저는 어떤 값을 전달받아서 그 값에 의해서 서로 다른 결과물을 구하게 됩니다.
- ❖ 값을 프로시저에 전달하기 위해서 프로시저 이름 다음에 괄호로 둘러싼 부분에 전달 받을 값을 저장할 변수를 기술합니다.
- ❖ 이를 ARGUMENT 우리나라 말로 매개 변수라 합니다.
- ❖ 프로시저는 매개 변수의 값에 따라 서로 다른 동작을 수행하게 됩니다.
- ❖ [MODE] 는 IN과 OUT, INOUT 세 가지를 기술할 수 있는데 IN 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용합니다. INOUT은 두 가지 목적에 모두 사용됩니다.

## 〈실습하기〉 저장 프로시저 생성하기

사원 테이블에 저장된 모든 사원을 삭제하는 프로시저를 작성해보도록 하겠습니다.

1. 모든 사원을 삭제하는 프로시저를 실행시키기 위해서 미리 사원 테이블을 복사해서 새로운 사원 테이블을 만들어 놓읍시다.
2. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하십시오.(파일이름:PROC01.sql)

```
CREATE OR REPLACE PROCEDURE DEL_ALL  
IS  
BEGIN  
DELETE FROM EMP01;  
END;  
/
```

3. 작성이 완료한 후에 파일을 저장합니다. SQL> 프롬프트에 @파일명을 입력하면 SQL 파일 내부에 기술한 PL/SQL 이 실행된 후 결과가 출력됩니다.

## 〈실습하기〉 저장 프로시저 생성하기

4. 생성된 저장 프로시저는 EXECUTE 명령어로 실행시킵니다.

```
EXECUTE DEL_ALL
```



## 1.1 저장 프로시저의 오류 원인 살피기



- ❖ ‘프로시저가 생성되었습니다.’ 란 메시지와 함께 한 번에 저장 프로시저를 성공적으로 생성할 수도 있지만, ‘경고: 컴파일 오류와 함께 프로시저가 생성되었습니다’ 와 같은 메시지가 출력되면 저장 프로시저를 생성하는 과정에서 오류가 발생해서 저장 프로시저 생성에 실패하는 경우입니다.

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

SQL> @PROC01

경고: 컴파일 오류와 함께 프로시저가 생성되었습니다.

SQL>
```

- ❖ 이럴 경우에는 오류를 제거해서 다시 저장프로시저를 생성해야 하는데, 오류 메시지를 모른 채 오류를 수정하기란 쉽지 않습니다.

## 1.1 저장 프로시저의 오류 원인 살피기



- ❖ 오류가 발생할 경우 “SHOW ERROR” 명령어를 수행하면 오류가 발생한 원인을 알 수 있게 됩니다.
- ❖ 원인을 분석하여 오류를 수정한 후 다시 저장 프로시저를 생성을 시도하여 ‘프로시저가 생성되었습니다.’란 메시지가 출력되어 저장 프로시저가 성공적으로 생성될 때까지 오류 수정 작업을 반복해야 합니다.

## <실습하기> 저장 프로시저 작성시 발생하는 오류 처리하기

저장 프로시저를 작성시 오류가 발생하면 이를 수정하기 위한 방법을 살펴봅시다.

1. PROC01.sql 파일을 열어서 다음과 같이 수정한 후 다시 ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하십시오.(파일이름:PROC01.sql)

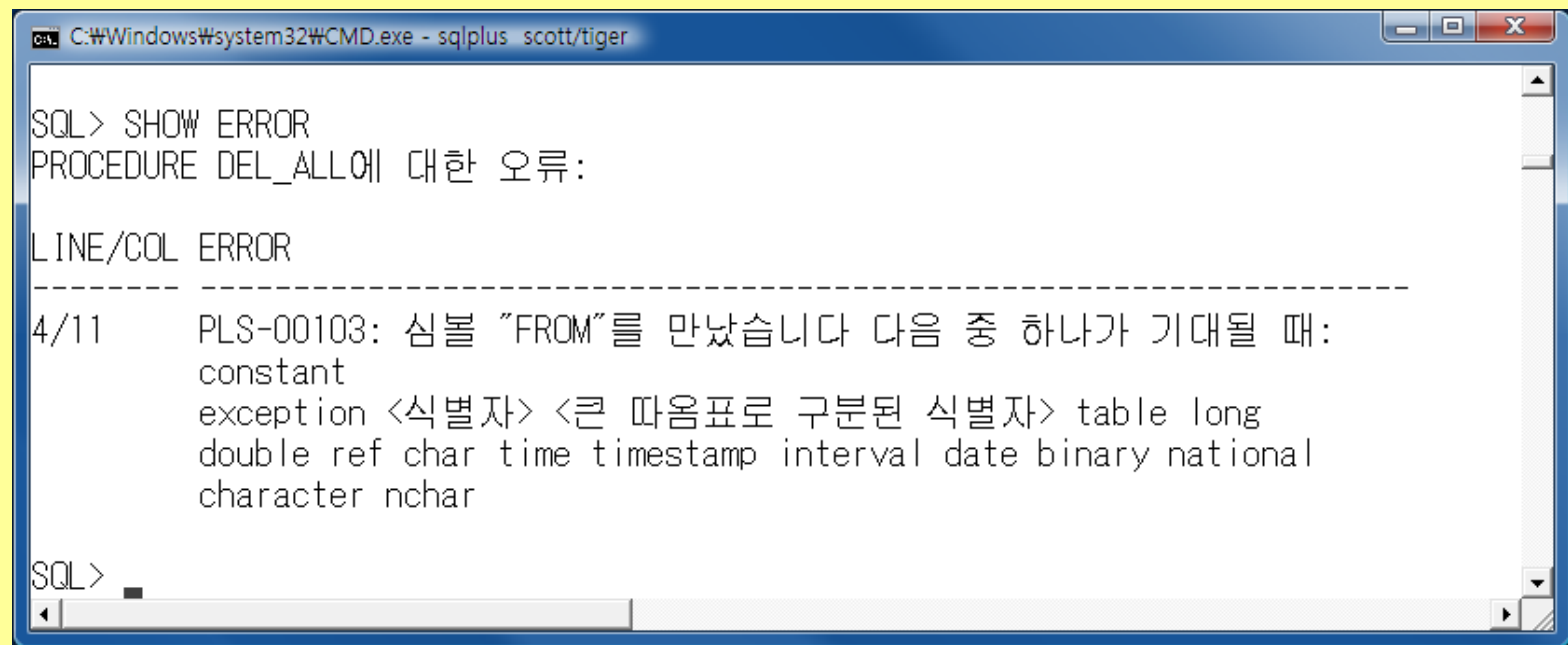
```
CREATE OR REPLACE PROCEDURE DEL_ALL  
IS  
DELETE FROM EMP01;  
END;  
/
```

2. SQL> 프롬프트에 @파일명을 입력하여 SQL 파일 내부에 기술한 PL/SQL을 실행하면 '경고: 컴파일 오류와 함께 프로시저가 생성되었습니다'와 같은 메시지가 출력됩니다.

## <실습하기> 저장 프로시저 작성시 발생하는 오류 처리하기

3. 발생한 오류에 대한 정보를 알고 싶을 때에는 SHOW ERROR를 입력합니다.

### SHOW ERROR



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

SQL> SHOW ERROR
PROCEDURE DEL_ALL에 대한 오류:

LINE/COL ERROR
-----
4/11      PLS-00103: 심볼 "FROM"를 만났습니다 다음 중 하나가 기대될 때:
          constant
          exception <식별자> <큰 따옴표로 구분된 식별자> table long
          double ref char time timestamp interval date binary national
          character nchar

SQL>
```

4. 내용을 수정한 후 프로시저를 재 생성하여 실행 시킵니다.

## 02. 저장 프로시저 조회하기



- ❖ 저장 프로시저를 작성한 후 사용자가 저장 프로시저가 생성되었는지 확인하려면 `USER_SOURCE` 살펴보면 됩니다.
- ❖ 다음은 `USER_SOURCE`의 구조입니다.

```
DESC USER_SOURCE
```

- ❖ `USER_SOURCE`의 내용을 조회하면 어떤 저장 프로시저가 생성되어 있는지와 해당 프로시저의 내용이 무엇인지 확인할 수 있습니다.

### 03. 저장 프로시저의 매개 변수



- ❖ DEL\_ALL 저장 프로시저는 사원 테이블의 모든 내용을 삭제합니다.
- ❖ 만일 특정 사원만을 삭제하려면 어떻게 해야 할까요?
- ❖ 저장 프로시저를 생성할 때 삭제하고자하는 사원의 이름이나 사원 번호를 프로시저에 전달해 주어 이와 일치하는 사원을 삭제하면 됩니다.
- ❖ 저장 프로시저에 값을 전달해 주기 위해서 매개 변수를 사용됩니다.

## 03. 저장 프로시저의 매개 변수



- ❖ 매개변수가 있는 저장프로시저는 다음과 같이 정의합니다.

```
CREATE OR REPLACE PROCEDURE  
DEL_ENAME(VENAME EMP01.ENAME%TYPE)  
IS  
BEGIN  
DELETE FROM EMP01 WHERE ENAME=VENAME;  
END;  
/
```

- ❖ 저장 프로시저 이름인 DEL\_ENAME 다음에 ( )를 추가하여 그 안에 선언한 변수가 매개 변수입니다. 이 매개변수에 값은 프로시저를 호출할 때 전달해 줍니다.

```
EXECUTE DEL_ENAME('SMITH');
```

## 04. IN, OUT, INOUT 매개 변수



- ❖ CREATE PROCEDURE로 프로시저를 생성할 때 MODE를 지정하여 매개변수를 선언할 수 있는데 MODE 에 IN, OUT, INOUT 세 가지를 기술할 수 있습니다.
- ❖ IN 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용합니다.
- ❖ INOUT은 두 가지 목적에 모두 사용됩니다.
- ❖ 이번에는 MODE를 지정하면 어떠한 기능이 부여되는지 자세히 살펴보기로 합니다.



## 4.1 IN 매개 변수



- ❖ 앞선 예제 중에서 매개변수로 사원의 이름을 전달받아서 해당 사원을 삭제하는 프로시저인 DEL\_ENAME를 작성해보았습니다.
- ❖ DEL\_ENAME 프로시저에서 사용된 매개변수는 프로시저를 호출할 때 기술한 값을 프로시저 내부에서 받아서 사용하고 있습니다.

```
EXECUTE DEL_ENAME('SMITH');
```



```
CREATE PROCEDURE DEL_ENAME(VENAME EMP01.ENAME%TYPE)
```

- ❖ 이렇게 프로시저 호출시 넘겨준 값을 받아오기 위한 매개변수는 MODE를 IN으로 지정해서 선언합니다.

```
CREATE PROCEDURE DEL_ENAME(VENAME IN EMP01.ENAME%TYPE)
```

## 4.2 OUT 매개 변수



- ❖ 프로시저에 구한 결과 값을 얻어 내기 위해서는 MODE를 OUT으로 지정합니다.
- ❖ 다음은 저장 프로시저를 생성할 때 IN, OUT 매개변수를 사용한 예입니다.

```
CREATE OR REPLACE PROCEDURE SEL_EMPNO  
( VEMPNO IN EMP.EMPNO%TYPE,  
  VENAME OUT EMP.ENAME%TYPE,  
  VSAL OUT EMP.SAL%TYPE,  
  VJOB OUT EMP.JOB%TYPE  
)
```

- ❖ 사원 번호로 특정 고객을 조회할 것이기 때문에 사원 번호를 IN으로 지정하고 조회해서 얻은 고객의 정보 중에서 고객의 이름과 급여와 담당 업무를 얻어오기 위해서 이름과 급여와 담당 업무컬럼을 OUT으로 지정하였습니다.

## 4.2 OUT 매개 변수



- ❖ 저장 프로시저 SEL\_EMPNO를 호출할 때에도 이전과 마찬가지로 EXECUTE 명령어로 실행시킵니다. OUT 매개변수에는 값을 받아오기 위해서는 프로시저 호출시 변수 앞에 ‘:’ 를 덧붙입니다.

```
EXECUTE SEL_EMPNO(7788, :VAR_ENAME, :VAR_SAL, :VAR_JOB)
```

- ❖ :를 덧붙여주는 변수는 미리 다음과 같이 선언되어 있어야 합니다.

```
VARIABLE VAR_ENAME VARCHAR2(15);  
VARIABLE VAR_SAL NUMBER;  
VARIABLE VAR_JOB VARCHAR2(9);
```

- ❖ 위와 같이 선언한 변수를 바인드 변수라고 합니다.

## 05. 저장 함수 생성



- ❖ 저장 함수는 저장 프로시저와 거의 유사한 용도로 사용됩니다.
- ❖ 차이점이라곤 함수는 실행 결과를 되돌려 받을 수 있다는 점입니다.
- ❖ 다음은 저장 함수를 만드는 기본 형식입니다.

```
CREATE [OR REPLACE ] FUNCTION function_name  
( argument1 [mode] data_taye,  
  argument2 [mode] data_taye . . .  
)  
IS  
RETURN data_type;  
BEGIN  
statement1;  
statement2;  
RETURN variable_name;  
END;
```

## 05. 저장 함수 생성



- ❖ 프로시저를 만들 때에는 PROCEDURE라고 기술하지만, 함수를 만들 때에는 FUNCTION이라고 기술합니다.
- ❖ 함수는 결과를 되돌려 받기 위해서 함수가 되돌려 받게 되는 자료형과 되돌려 받을 값을 기술해야 합니다.
- ❖ 저장 함수는 호결과를 얻어오기 위해서 호출 방식에 있어서도 저장 프로시저와 차이점이 있습니다.

```
EXECUTE :variable_name := function_name(argument_list);
```

## 〈실습하기〉 저장함수 작성하기

다음은 특별 보너스를 지급하기 위한 저장 함수를 작성해 봅시다. 보너스는 급여의 200%를 지급한다고 합시다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하십시오.(파일이름:PROC04.SQL)

```
CREATE OR REPLACE FUNCTION CAL_BONUS(  
    VEMPNO IN EMP.EMPNO%TYPE )  
    RETURN NUMBER  
IS  
    VSAL NUMBER(7, 2);  
BEGIN  
    SELECT SAL INTO VSAL  
    FROM EMP  
    WHERE EMPNO = VEMPNO;  
  
    RETURN (VSAL * 200);  
END;  
/
```

## 〈실습하기〉 저장함수 작성하기

2. 함수의 결과 값을 저장할 변수를 선언한 후에 아래와 같이 함수 호출합니다.

```
VARIABLE VAR_RES NUMBER;  
EXECUTE :VAR_RES := CAL_BONUS(7788);
```

## 06. 커서



- ❖ 앞선 PL/SQL 예제에서는 처리 결과가 1개의 행인 SELECT 문만을 다루었습니다.
- ❖ 하지만 대부분의 SELECT 문은 수행 후 반환되는 행의 개수가 한 개 이상입니다.
- ❖ 처리 결과가 여러 개의 행으로 구해지는 SELECT 문을 처리하려면 지금부터 학습할 커서를 이용해야 합니다.



## 05. 커서



**DECLARE**

-- 커서 선언

**CURSOR** *cursor\_name* IS *statement*;

**BEGIN**

-- 커서 열기

**OPEN** *cursor\_name*;

--커서로부터 데이터를 읽어와 변수에 저장

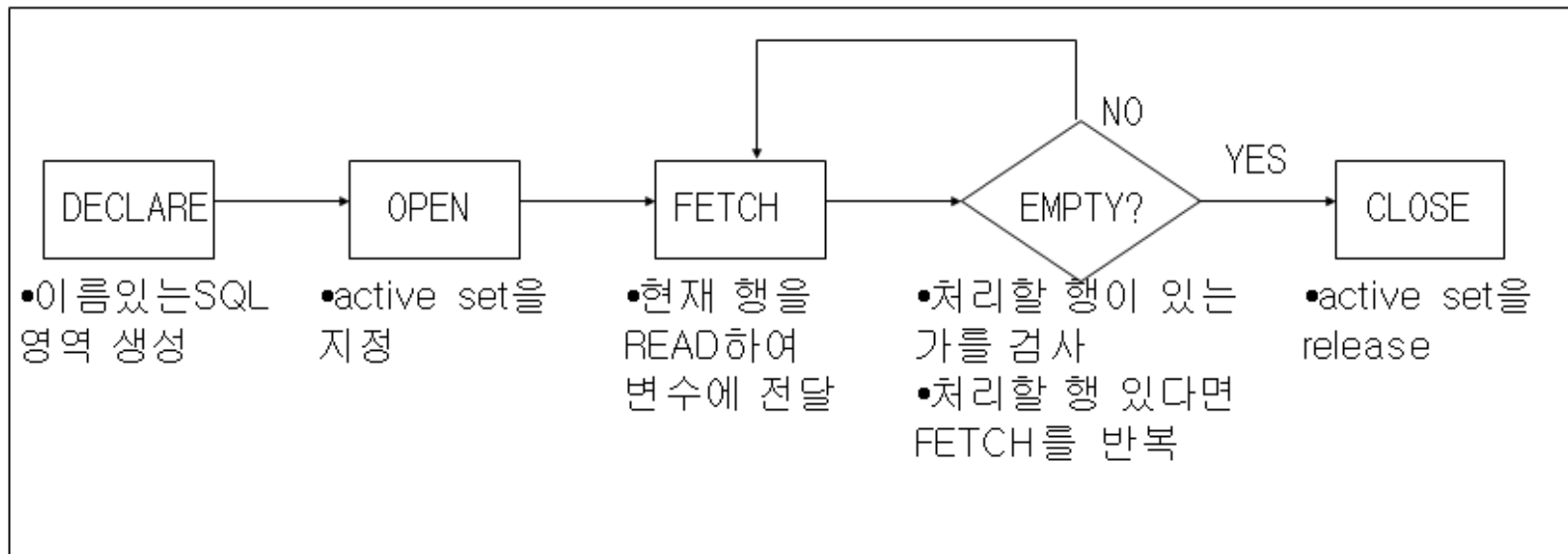
**FETCH** *cur\_name* INTO *variable\_name*;

--커서 닫기

**CLOSE** *cursor\_name*;

**END;**

## 05. 커서



# DECLARE CURSOR



- ❖ 명시적으로 CURSOR를 선언하기 위해 CURSOR문장을 사용합니다.

```
CURSOR cursor_name IS  
select_statement;
```

구문	의미
<b>cursor_name</b>	PL/SQL 식별자
<b>select_statement</b>	INTO절이 없는 SELECT 문장

# OPEN CURSOR



- ❖ 질의를 수행하고 검색 조건을 충족하는 모든 행으로 구성된 결과 셋을 생성하기 위해 CURSOR를 OPEN합니다. CURSOR는 이제 결과 셋에서 첫 번째 행을 가리킨다.

```
OPEN cursor_name;
```

- ❖ 부서 테이블의 모든 내용을 조회하는 SELECT문과 연결된 커서 C1을 오픈합니다.

```
OPEN C1;
```

- ❖ C1을 오픈하면 검색 조건에 만족하는 모든 행으로 구성된 결과 셋이 구해지고 부서 테이블의 첫 번째 행을 가리키게 됩니다.

# FETCH CURSOR



- ❖ **FETCH 문은 결과 셋에서 로우 단위로 데이터를 읽어 들인다. 각 인출(FETCH) 후에 CURSOR는 결과 셋에서 다음 행으로 이동합니다.**

```
FETCH cursor_name INTO {variable1[,variable2, . . . .]};
```

- ❖ **FETCH 문장은 현재 행에 대한 정보를 얻어 와서 INTO 뒤에 기술한 변수에 저장한 후 다음 행으로 이동합니다. 얻어진 여러 개의 로우에 대한 결과값을 모두 처리하려면 반복문에 FETCH 문을 기술해야 합니다.**

```
LOOP  
FETCH C1 INTO VDEPT.DEPTNO, VDEPT.DNAME, VDEPT.LOC;  
EXIT WHEN C1%NOTFOUND;  
END LOOP;
```

# FETCH CURSOR



- ❖ 커서가 끝에 위치하게 되면 반복문을 탈출해야 합니다.
- ❖ 단순 LOOP는 내부에 EXIT WHEN 문장을 포함하고 있다가 EXIT WHEN 다음에 기술한 조건에 만족하면 단순 LOOP를 탈출하게 됩니다.
- ❖ 반복문을 탈출할 조건으로 “C1%NOTFOUND “를 기술하였습니다.
- ❖ NOTFOUND는 커서의 상태를 알려주는 속성 중에 하나인데 커서 영역의 자료가 모두 FETCH됐다면 TRUE를 되돌립니다.
- ❖ 커서 C1영역의 자료가 모두 FETCH 되면 반복문을 탈출하게 됩니다.

## 커서의 상태



- ❖ **FETCH 문을 설명하면서 커서의 속성 중에 NOTFOUND를 언급하였는데 오라클에서는 이외에도 다양한 커서의 속성을 통해 커서의 상태를 알려주는데 이 속성을 이용해서 커서를 제어해야 합니다.**

속성	의미
%NOTFOUND	커서 영역의 자료가 모두 FETCH됐었다면 TRUE
%FOUND	커서 영역에 FETCH 되지 않은 자료가 있다면 TRUE
%ISOPEN	커서가 OPEN된 상태이면 TRUE
%ROWCOUNT	커서가 얻어 온 레코드의 개수

# CLOSE CURSOR



- ❖ **CLOSE문장은 CURSOR를 사용할 수 없게 하고 결과 셋의 정의를 해제합니다.**
- ❖ **SELECT 문장이 다 처리된 완성 후에는 CURSOR를 닫는다. 필요하다면 CURSOR를 다시 열수도 있습니다.**

```
CLOSE cursor_name;
```



## 〈실습하기〉 부서 테이블의 모든 내용을 조회하기

커서를 사용하여 부서 테이블의 모든 내용을 출력합니다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하십시오.(파일이름:PROC05.SQL)

```
CREATE OR REPLACE PROCEDURE CURSOR_SAMPLE01
IS
    VDEPT DEPT%ROWTYPE;
    CURSOR C1
    IS
        SELECT *      FROM  DEPT;
BEGIN
    DBMS_OUTPUT.PUT_LINE(' 부서번호 / 부서명 / 지역명 ');
    DBMS_OUTPUT.PUT_LINE(' ----- ');

    OPEN C1;
```

## 〈실습하기〉 부서 테이블의 모든 내용을 조회하기

LOOP

FETCH C1 INTO VDEPT.DEPTNO, VDEPT.DNAME, VDEPT.LOC

EXIT WHEN C1%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(VDEPT.DEPTNO ||  
                  '      ' || VDEPT.DNAME || '      ' || VDEPT.LOC);

END LOOP;

CLOSE C1;

END;

/

## 6.1 CURSOR와 FOR LOOP



- ❖ CURSOR FOR LOOP는 명시적 CURSOR에서 행을 처리합니다. LOOP에서 각 반복마다 CURSOR를 열고 행을 인출(FETCH)하고 모든 행이 처리되면 자동으로 CURSOR가 CLOSE되므로 사용하기가 편리합니다.

```
FOR record_name IN cursor_name LOOP  
  statement1;  
  statement2;  
  . . .  
END LOOP
```

- ❖ OPEN ~ FETCH ~ CLOSE가 없이 FOR ~ LOOP ~ END LOOP문을 사용하여 보다 간단하게 커서를 처리해 봅시다.

## 〈실습하기〉 부서 테이블의 모든 내용 출력하기

커서를 사용하여 부서 테이블의 모든 내용 출력해 보도록 합시다.

1. ED 다음에 파일이름을 입력하여 새로 생긴 SQL 파일에 다음과 같이 입력하십시오.(파일이름:PROC06.SQL)

```
CREATE OR REPLACE PROCEDURE CURSOR_SAMPLE02
IS
VDEPT DEPT%ROWTYPE;
CURSOR C1
IS
SELECT * FROM DEPT;
BEGIN
DBMS_OUTPUT.PUT_LINE('부서번호 / 부서명 / 지역명');
DBMS_OUTPUT.PUT_LINE('-----');
FOR VDEPT IN C1 LOOP
EXIT WHEN C1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(VDEPT.DEPTNO ||
' ' || VDEPT.DNAME || ' ' || VDEPT.LOC);
END LOOP;
END;
/
```



# Thank You !

Dynamic\_오라클 11g + PL/SQL 입문 21장