

PROCEDURE & FUNCTION

▶ PROCEDURE

PL/SQL문을 저장하는 객체로

필요할 때마다 복잡한 구문을 다시 입력할 필요 없이 간단하게 호출해서 실행 결과를 얻을 수 있음

✓ 예시

```
CREATE TABLE EMP_DUP  
AS SELECT * FROM EMPLOYEE;
```


Table EMP_DUP이 (가) 생성되었습니다.

```
CREATE OR REPLACE PROCEDURE DEL_ALL_EMP  
IS  
BEGIN
```

```
    DELETE FROM EMP_DUP;  
    COMMIT;
```

```
END;  
/  
Procedure DEL_ALL_EMP이 (가) 컴파일되었습니다.
```

```
SELECT * FROM EMP_DUP;
```

 SQL | 인출된 모든 행: 24(0.015초)

```
EXEC DEL_ALL_EMP;
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

```
SELECT * FROM EMP_DUP;
```

 SQL | 인출된 모든 행: 0(0초)

► PROCEDURE

DESC USER_SOURCE;

이름	날	유형
NAME	VARCHAR2 (30)	
TYPE	VARCHAR2 (12)	
LINE	NUMBER	
TEXT	VARCHAR2 (4000)	

SELECT * FROM USER_SOURCE;

	NAME	TYPE	LINE	TEXT
1	DEL_ALL_EMP	PROCEDURE	1	PROCEDURE DEL_ALL_EMP
2	DEL_ALL_EMP	PROCEDURE	2	IS
3	DEL_ALL_EMP	PROCEDURE	3	BEGIN
4	DEL_ALL_EMP	PROCEDURE	4	DELETE FROM EMP_DUP;
5	DEL_ALL_EMP	PROCEDURE	5	
6	DEL_ALL_EMP	PROCEDURE	6	COMMIT;
7	DEL_ALL_EMP	PROCEDURE	7	END;


▶ PROCEDURE

✓ 매개변수 있는 프로시저

```
CREATE OR REPLACE PROCEDURE DEL_EMP_ID
(V_EMP_ID EMPLOYEE.EMP_ID%TYPE)
IS
BEGIN
    DELETE FROM EMPLOYEE
    WHERE EMP_ID = V_EMP_ID;
    COMMIT;
END;
/
```

Procedure DEL_EMP_ID이 (가) 컴파일되었습니다.

```
SELECT * FROM EMPLOYEE;
```

 SQL | 인출된 모든 행: 24(0,004초)

```
EXECUTE DEL_EMP_ID('&EMP_ID');
```


대체 변수 입력

EMP_ID에 대한 값 입력:

확인 취소

PL/SQL 프로시저가 성공적으로 완료되었습니다.

```
SELECT * FROM EMPLOYEE;
```

 SQL | 인출된 모든 행: 23(0,003초)

▶ PROCEDURE

✓ IN/OUT 매개변수 있는 프로시저

```
CREATE OR REPLACE PROCEDURE SELECT_EMP_ID(  
    V_EMP_ID IN EMPLOYEE.EMP_ID%TYPE,  
    V_EMP_NAME OUT EMPLOYEE.EMP_NAME%TYPE,  
    V_SALARY OUT EMPLOYEE.SALARY%TYPE,  
    V_BONUS OUT EMPLOYEE.BONUS%TYPE  
)  
IS  
BEGIN  
    SELECT EMP_NAME, SALARY, NVL(BONUS, 0)  
    INTO V_EMP_NAME, V_SALARY, V_BONUS  
    FROM EMPLOYEE  
    WHERE EMP_ID = V_EMP_ID;  
END;  
/
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

▶ PROCEDURE

✓ IN/OUT 매개변수 있는 프로시저

```
VARIABLE VAR_EMP_NAME VARCHAR2(30);
```

```
VARIABLE VAR_SALARY NUMBER;
```

```
VARIABLE VAR_BONUS NUMBER;
```

* 바인드 변수 선언

PL/SQL 프로시저가 성공적으로 완료되었습니다.

```
EXEC SELECT_EMPID(200, :VAR_EMP_NAME, :VAR_SALARY, :VAR_BONUS);
```

```
PRINT VAR_EMP_NAME;
```

```
PRINT VAR_SALARY;
```

```
PRINT VAR_BONUS;
```

```
VAR_EMP_NAME
```

```
----
```

```
선동일
```

```
VAR_SALARY
```

```
-----
```

```
8000000
```

```
VAR_BONUS
```

```
----
```

```
0.3
```

* 프로시저 실행과 동시에 모든 바인딩 변수를 출력하기 위해선 SET AUTOPRINT ON;을 실행 시켜야 함

▶ FUNCTION

프로시저와 거의 유사한 용도로 사용하지만 실행결과를 되돌려 받을 수 있다는 점에서 프로시저와 다름

✓ 예시

```
CREATE OR REPLACE FUNCTION BONUS_CALC(V_EMPID EMPLOYEE.EMP_ID%TYPE)
```

```
RETURN NUMBER
```

```
IS
```

```
    V_SAL EMPLOYEE.SALARY%TYPE;  
    V_BONUS EMPLOYEE.BONUS%TYPE;  
    CALC_SAL NUMBER;
```

```
BEGIN
```

```
    SELECT SALARY, NVL(BONUS, 0)  
    INTO V_SAL, V_BONUS  
    FROM EMPLOYEE  
    WHERE EMP_ID = V_EMPID;
```

```
    CALC_SAL := (V_SAL + (V_SAL + V_BONUS)) * 12;  
    RETURN CALC_SAL;
```

```
END;
```

```
/
```

Function BONUS_CALC이(가) 컴파일되었습니다.

```
VARIABLE VAR_CALC NUMBER;  
EXEC :VAR_CALC =  
        BONUS_CALC('&EMP_ID');
```

대체 변수 입력

EMP_ID에 대한 값 입력::

확인 취소

PL/SQL 프로시저가 성공적으로 완료되었습니다.

```
CALC_SAL  
-----  
72000000
```

▶ FUNCTION

✓ 예시

```
SELECT EMP_ID, EMP_NAME,  
       BONUS_CALC(EMP_ID)  
FROM EMPLOYEE;
```

SQL | 인출된 모든 행: 23(0초)

	EMP_ID	EMP_NAME	BONUS_CALC(EMP_ID)
1	200	선동일	124800000
2	201	송종기	72000000
3	202	노웅철	44400000
4	203	송은희	33600000
5	204	유재식	48960000
6	205	정중하	46800000
7	206	박나라	21600000
8	207	하미유	29040000
9	208	김해술	30000000
10	209	심봉선	48300000
11	210	윤은혜	24000000
12	211	전형돈	24000000
13	212	장프위	38250000
14	213	하동운	30624000
15	214	방영수	16560000
16	215	대복혼	45120000
17	216	차태연	40032000
18	217	전지연	57096000
19	218	미오리	34680000
20	219	임시환	18600000
21	220	이중석	29880000
22	221	유하진	29760000
23	222	이태림	39467088

```
SELECT EMP_ID, EMP_NAME,  
       BONUS_CALC(EMP_ID)  
FROM EMPLOYEE  
WHERE BONUS_CALC(EMP_ID) >= 3000000  
ORDER BY 3 DESC;
```

SQL | 인출된 모든 행: 15(0초)

	EMP_ID	EMP_NAME	BONUS_CALC(EMP_ID)
1	200	선동일	124800000
2	201	송종기	72000000
3	217	전지연	57096000
4	204	유재식	48960000
5	209	심봉선	48300000
6	205	정중하	46800000
7	215	대복혼	45120000
8	202	노웅철	44400000
9	216	차태연	40032000
10	222	이태림	39467088
11	212	장프위	38250000
12	218	미오리	34680000
13	203	송은희	33600000
14	213	하동운	30624000
15	208	김해술	30000000

▶ CURSOR

결과가 여러 개의 행으로 구해지는 SELECT문을 처리하기 위해 실행 결과를 저장해놓은 객체

CURSOR ~ OPEN ~ FETCH ~ CLOSE 단계로 진행

✓ 상태

속 성	설 명
%NOTFOUND	커서 영역의 자료가 모두 FETCH되어 다음 영역이 존재하지 않으면 TRUE
%FOUND	커서 영역에 아직 FETCH되지 않은 자료가 있으면 TRUE
%ISOPEN	커서가 OPEN된 상태이면 TRUE
%ROWCOUNT	커서가 얻어 온 레코드의 개수

▶ CURSOR

✓ 예시1_1

```
CREATE OR REPLACE PROCEDURE CURSOR_DEPT
IS
```

```
    V_DEPT DEPARTMENT%ROWTYPE;
```

```
    CURSOR C1
```

```
    IS
```

```
        SELECT * FROM DEPARTMENT;
```

```
    BEGIN
```

```
        OPEN C1;
```

```
        LOOP
```

```
            FETCH C1 INTO V_DEPT.DEPT_ID, V_DEPT.DEPT_TITLE,
                          V_DEPT.LOCATION_ID;
```

```
            EXIT WHEN C1%NOTFOUND;
```

```
            DBMS_OUTPUT.PUT_LINE('부서코드 : ' || V_DEPT.DEPT_ID ||
                                  ' , 부서명 : ' || V_DEPT.DEPT_TITLE ||
                                  ' , 지역 : ' || V_DEPT.LOCATION_ID);
```

```
        END LOOP;
```

```
    CLOSE C1;
```

```
END;
```

```
/
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

부서코드 : D1 , 부서명 : 인사관리부 , 지역 : L1

부서코드 : D2 , 부서명 : 회계관리부 , 지역 : L1

부서코드 : D3 , 부서명 : 마케팅부 , 지역 : L1

부서코드 : D4 , 부서명 : 국내영업부 , 지역 : L1

부서코드 : D5 , 부서명 : 해외영업1부 , 지역 : L2

부서코드 : D6 , 부서명 : 해외영업2부 , 지역 : L3

부서코드 : D7 , 부서명 : 해외영업3부 , 지역 : L4

부서코드 : D8 , 부서명 : 기술지원부 , 지역 : L5

부서코드 : D9 , 부서명 : 총무부 , 지역 : L1

▶ CURSOR

✓ 예시1_2

```
CREATE OR REPLACE PROCEDURE CURSOR_DEPT
IS
```

```
    V_DEPT DEPARTMENT%ROWTYPE;
```

```
    CURSOR C1
```

```
    IS
```

```
        SELECT * FROM DEPARTMENT;
```

```
    BEGIN
```

```
        FOR V_DEPT IN C1 LOOP
```

```
            DBMS_OUTPUT.PUT_LINE('부서코드 : ' || V_DEPT.DEPT_ID ||  
                                ', 부서명 : ' || V_DEPT.DEPT_TITLE ||  
                                ', 지역 : ' || V_DEPT.LOCATION_ID);
```

```
        END LOOP;
```

```
    END;
```

```
    /
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

```
부서코드 : D1 , 부서명 : 인사관리부 , 지역 : L1  
부서코드 : D2 , 부서명 : 회계관리부 , 지역 : L1  
부서코드 : D3 , 부서명 : 마케팅부 , 지역 : L1  
부서코드 : D4 , 부서명 : 국내영업부 , 지역 : L1  
부서코드 : D5 , 부서명 : 해외영업1부 , 지역 : L2  
부서코드 : D6 , 부서명 : 해외영업2부 , 지역 : L3  
부서코드 : D7 , 부서명 : 해외영업3부 , 지역 : L4  
부서코드 : D8 , 부서명 : 기술지원부 , 지역 : L5  
부서코드 : D9 , 부서명 : 총무부 , 지역 : L1
```

* FOR IN LOOP를 이용하면 LOOP 반복 시 자동으로 CURSOR를 OPEN하고 행 인출(FETCH)
LOOP 종료 시 자동으로 CURSOR CLOSE

▶ CURSOR

✓ 예시1_3

```
CREATE OR REPLACE PROCEDURE CURSOR_DEPT
IS
    V_DEPT DEPARTMENT%ROWTYPE;
BEGIN
    FOR V_DEPT IN (SELECT * FROM DEPARTMENT) LOOP
        DBMS_OUTPUT.PUT_LINE('부서코드 : ' || V_DEPT.DEPT_ID
                               || ', 부서명 : ' || V_DEPT.DEPT_TITLE
                               || ', 지역 : ' || V_DEPT.LOCATION_ID);
    END LOOP;
END;
```

* FOR IN문을 사용하면 커서의 선언도 생략 가능

PL/SQL 프로시저가 성공적으로 완료되었습니다.

```
부서코드 : D1 , 부서명 : 인사관리부 , 지역 : L1
부서코드 : D2 , 부서명 : 회계관리부 , 지역 : L1
부서코드 : D3 , 부서명 : 마케팅부 , 지역 : L1
부서코드 : D4 , 부서명 : 국내영업부 , 지역 : L1
부서코드 : D5 , 부서명 : 해외영업1부 , 지역 : L2
부서코드 : D6 , 부서명 : 해외영업2부 , 지역 : L3
부서코드 : D7 , 부서명 : 해외영업3부 , 지역 : L4
부서코드 : D8 , 부서명 : 기술지원부 , 지역 : L5
부서코드 : D9 , 부서명 : 총무부 , 지역 : L1
```