

15. 저장 프로시저 • 패키지 • 트리거 • 함수

- 테이블에서 데이터를 조회해 원하는대로 조작하고 그 결과를 다른 테이블에 다시 저장하거나 수정하는 일련의 처리를 할 때 주로 프로시저를 사용하는데 그 프로시저에 대해 학습한다.
- 관련 있는 프로시저를 보다 효율적으로 관리하기 위해서 패키지 단위로 배포하기 위한 패키지를 패키지를 학습한다 .
- 특정 테이블의 데이터에 변경이 가해졌을 때 자동으로 수행되는 트리거를 학습한다.

1) 저장 프로시저(Stored Procedure : 스토어드 프로시저)

저장 프로시저는 자주 사용되는 쿼리문을 모듈화시켜서 필요할 때마다 호출하여 사용하는 것을 말한다.

프로시저의 생성구문

먼저 생성된 프로시저를 변경하거나 재 생성하는 옵션

CREATE [OR REPLACE] PROCEDURE procedure_name

(매개변수1 [mode] 자료형,
매개변수2 [mode] 자료형 ...)

MODE에 따라 프로시저를 호출 시 값을 전달 받거나 프로시저 호출 후 해당 매개 변수 값을 받아 사용

IS

local_variable declaration

BEGIN

statement1;

IN(입력) / OUT(출력) / IN OUT(입력과 출력을 동시에 한다)

END [procedure_name];

CREATE OR REPLACE PROCEDURE EMPPROC

IS

VWORD VARCHAR2(1);

VEMP EMPLOYEES%ROWTYPE;

CURSOR C1 (VWORD VARCHAR2)

IS

SELECT EMPLOYEE_ID, FIRST_NAME, SALARY

FROM EMPLOYEES

WHERE FIRST_NAME LIKE '%' || VWORD || '%';

BEGIN

VWORD := DBMS_RANDOM.STRING('U',1);

DBMS_OUTPUT.PUT_LINE('임의의 문자 : ' || VWORD);

OPEN C1(VWORD);

DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여');

DBMS_OUTPUT.PUT_LINE('-----');

LOOP

FETCH C1 INTO VEMP.EMPLOYEE_ID, VEMP.FIRST_NAME, VEMP.SALARY;

IF C1%ROWCOUNT = 0 THEN

```

        DBMS_OUTPUT.PUT_LINE('해당 직원이 존재하지 않습니다') ;
    END IF;
    EXIT WHEN C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' || VEMP.SALARY);
END LOOP;
END;
/

```

생성된 저장프로시저는 **EXECUTE 프로시저명** 또는 **EXEC 프로시저명** 명령어로 실행시킨다.

```
EXECUTE EMPPROC;
```



저장 프로시저를 작성한 후 사용자가 저장 프로시저가 생성되었는지 확인하려면 **USER_SOURCE** 살펴보면 된다.

```

SELECT *
FROM USER_SOURCE;

```

NAME	TYPE	LINE	TEXT
ADD_JOB_HISTORY	PROCEDURE	13	END add_job_history;
EMPPROC	PROCEDURE	1	PROCEDURE EMPPROC
EMPPROC	PROCEDURE	2	IS
EMPPROC	PROCEDURE	3	VWORD VARCHAR2(1);
EMPPROC	PROCEDURE	4	VEMP EMPLOYEES%ROWTYPE;
EMPPROC	PROCEDURE	5	CURSOR C1 (VWORD VARCHAR2)
EMPPROC	PROCEDURE	6	IS
EMPPROC	PROCEDURE	7	SELECT EMPLOYEE_ID, FIRST_NAME, SALARY
EMPPROC	PROCEDURE	8	FROM EMPLOYEES
EMPPROC	PROCEDURE	9	WHERE FIRST_NAME LIKE '%' VWORD '%';
EMPPROC	PROCEDURE	10	BEGIN
EMPPROC	PROCEDURE	11	VWORD := DBMS_RANDOM.STRING('U',1);
EMPPROC	PROCEDURE	12	DBMS_OUTPUT.PUT_LINE('임의의 문자 : ' VWORD);
EMPPROC	PROCEDURE	13	OPEN C1(VWORD);
EMPPROC	PROCEDURE	14	DBMS_OUTPUT.PUT_LINE('직원번호 / 사원명 / 급여');
EMPPROC	PROCEDURE	15	DBMS_OUTPUT.PUT_LINE('-----');
EMPPROC	PROCEDURE	16	LOOP

SHOW ERROR를 입력하면 상세한 오류 메시지가 출력되므로 이 내용을 보고 오류를 수정하면 된다. 아

래 예제는 위의 프로서저에서 명령문 하나의 ";"를 생략하여 오류를 발생시킨 것이다. 이때 오류 메시지를 SHOW ERROR 메시지로 확인하여 보자.

```
CREATE OR REPLACE PROCEDURE EMPPROC
IS
  VWORD VARCHAR2(1);
  VEMP EMPLOYEES%ROWTYPE;
  CURSOR C1 (VWORD VARCHAR2)
  IS
    SELECT EMPLOYEE_ID, FIRST_NAME, SALARY
    FROM EMPLOYEES
    WHERE FIRST_NAME LIKE '%' || VWORD || '%';
BEGIN
  VWORD := DBMS_RANDOM.STRING('U',1);
  DBMS_OUTPUT.PUT_LINE('임의의 문자 : ' || VWORD);
  OPEN C1(VWORD);
  DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여')
  DBMS_OUTPUT.PUT_LINE('-----');
  LOOP
    FETCH C1 INTO VEMP.EMPLOYEE_ID, VEMP.FIRST_NAME, VEMP.SALARY;
    IF C1%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('해당 직원이 존재하지 않습니다') ;
    END IF;
    EXIT WHEN C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' || VEMP.SALARY);
  END LOOP;
END;
/
SHOW ERROR;
```

DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여')

여기에 ; 누락으로 에러발생.

전체를 선택하고 실행한다.

```

1 CREATE OR REPLACE PROCEDURE EMPPROC
2 IS
3     VWORD VARCHAR2(1);
4     VEMP EMPLOYEES%ROWTYPE;
5     CURSOR C1 (VWORD VARCHAR2)
6     IS
7     SELECT EMPLOYEE_ID, FIRST_NAME, SALARY
8     FROM EMPLOYEES
9     WHERE ENAME LIKE '%' || VWORD || '%';
10 BEGIN
11     VWORD := DBMS_RANDOM.STRING('U',1);
12     DBMS_OUTPUT.PUT_LINE('임의의 문자 : ' || VWORD);
13     OPEN C1(VWORD);
14     DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여')
15     DBMS_OUTPUT.PUT_LINE('-----');
16     LOOP
17         FETCH C1 INTO VEMP.EMPLOYEE_ID, VEMP.FIRST_NAME, VEMP.SALARY;
18         IF C1%ROWCOUNT = 0 THEN
19             DBMS_OUTPUT.PUT_LINE('해당 직원이 존재하지 않습니다') ;
20             END IF;
21             EXIT WHEN C1%NOTFOUND;
22             DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' || VEMP.SALARY);
23     END LOOP;
24 END;
25 /
26 SHOW ERROR;

```

스크립트 실행 결과

작업이 완료되었습니다. (0.086초)

PROCEDURE HR.EMPPROC에 대한 오류:

LINE/COL ERROR

15/5 PLS-00103: 실행 "DBMS_OUTPUT"를 만났습니다 다음 중 하나가 기대될 때:

:= . (% ;

실행이 ":"= 계속하기 위하여 "DBMS_OUTPUT"로 치환되었습니다

① 매개 변수

매개 변수는 프로시저 이름 뒤에 ()를 기술하여 그 내부에 매개변수를 정의한다.

형식은 **변수명 모드 자료형**으로 기술한다.

```

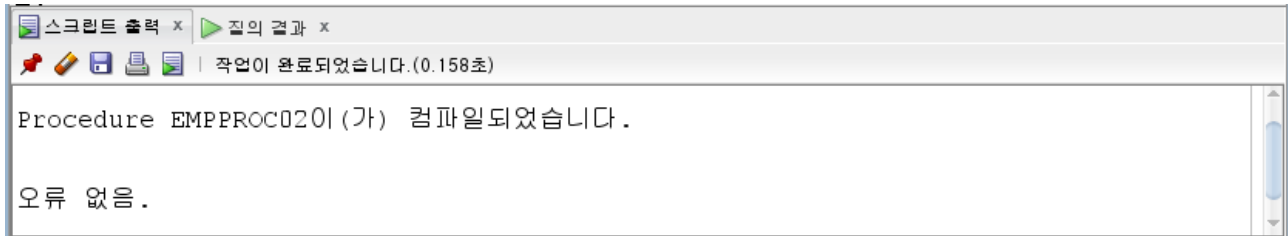
CREATE OR REPLACE PROCEDURE EMPPROC02
( VDEPTNO IN EMPLOYEES.DEPARTMENT_ID%TYPE )
IS
CURSOR C1
IS
SELECT * FROM EMPLOYEES
WHERE DEPARTMENT_ID=VDEPTNO;
BEGIN
    DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여');
    DBMS_OUTPUT.PUT_LINE('-----');
    FOR VEMP IN C1 LOOP

```

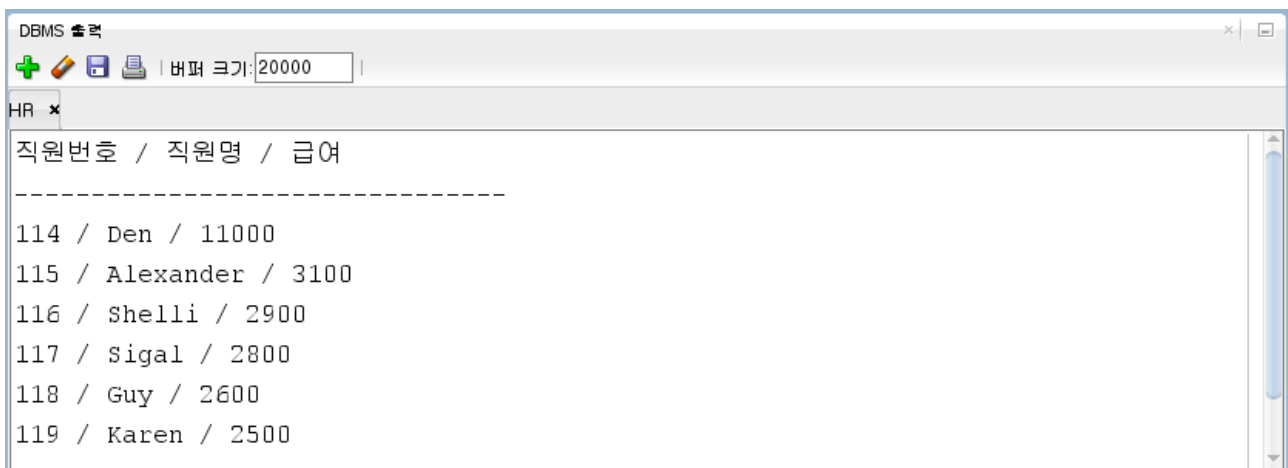
```

DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' || VEMP.SALARY);
END LOOP;
END;
/
SHOW ERROR;

```



```
EXECUTE EMPPROC02(30);
```



• IN MODE 매개변수

실행환경에서 서브 프로시저로 값을 전달한다.

부서별로 SALARY 인상. 부서코드가 10이면 10% 인상, 20이면 20% 인상, 나머지는 동결하는 쿼리문을 작성하여 보자. 그 전에 변경 전 데이터를 확인한다.


```

DROP TABLE EMP01;

CREATE TABLE EMP01
AS
SELECT DEPARTMENT_ID, FIRST_NAME, SALARY
FROM EMPLOYEES;

SELECT DEPARTMENT_ID, FIRST_NAME, SALARY
FROM EMP01
WHERE DEPARTMENT_ID IN(10, 20);

```



	DEPARTMENT_ID	FIRST_NAME	SALARY
1	10	Jennifer	4400
2	20	Michael	13000
3	20	Pat	6000

```

CREATE OR REPLACE PROCEDURE EMPPROC_INMODE
(V_DEPTNO IN EMP01.DEPARTMENT_ID%TYPE)
IS
BEGIN
    UPDATE EMP01 SET SALARY = DECODE(V_DEPTNO, 10, SALARY*1.1, 20, SALARY*1.2, SALARY)
    WHERE DEPARTMENT_ID = V_DEPTNO;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('수정이 완료되었습니다.');
```

```

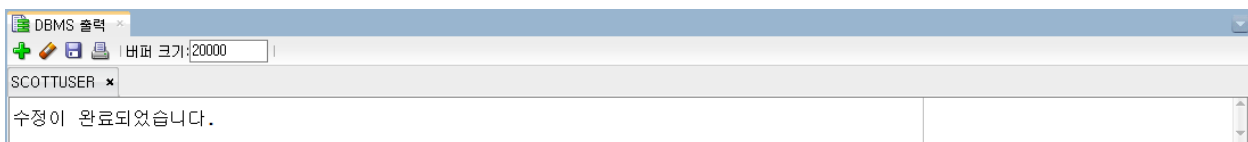
END EMPPROC_INMODE;
/
SHOW ERROR;
```



PROCEDURE EMPPROC_INMODE이 (가) 컴파일되었습니다.
오류가 없습니다.

```

EXECUTE EMPPROC_INMODE(10);
EXECUTE EMPPROC_INMODE(20);
```



수정이 완료되었습니다.

```

SELECT DEPARTMENT_ID, FIRST_NAME, SALARY
FROM EMP01
WHERE DEPARTMENT_ID IN(10, 20);
```



	DEPARTMENT_ID	FIRST_NAME	SALARY
1	10	Jennifer	4840
2	20	Michael	15600
3	20	Pat	7200

DEPT01 테이블을 생성한다.(이전 테이블은 삭제한다.)

컬럼명	자료형	크기	제약사항
DEPTNO	NUMBER	2	기본키
DNAME	VARCHAR2	14	NULL 허용하지 않음
LOC	VARCHAR2	13	NULL 허용하지 않음

```
DROP TABLE DEPT01;
```

```
CREATE TABLE DEPT01(
    deptno NUMBER(2) PRIMARY KEY,
    dname VARCHAR2(14) NOT NULL,
    loc VARCHAR2(13) NOT NULL
);
```

테이블 생성 후 아래의 결과를 얻을 수 있도록 데이터를 입력한다.

```
SELECT DEPTNO, DNAME, LOC
FROM DEPT01
ORDER BY DEPTNO;
```

	DEPTNO	DNAME	LOC
1	10	인사과	서울
2	20	총무과	대전
3	30	교육팀	서울
4	40	기술팀	인천
5	50	시설관리팀	광주

그리고 DEPT01 테이블에 CREDATE라는 이름의 컬럼을 날짜 자료형으로 추가하자.

```
ALTER TABLE DEPT01
ADD(CREDATE DATE DEFAULT SYSDATE);
```

```
SELECT DEPTNO, DNAME, LOC, CREDATE
FROM DEPT01
ORDER BY DEPTNO;
```

	DEPTNO	DNAME	LOC	CREDATE
1	10	인사과	서울	16/10/12
2	20	총무과	대전	16/10/12
3	30	교육팀	서울	16/10/12
4	40	기술팀	인천	16/10/12
5	50	시설관리팀	광주	16/10/12

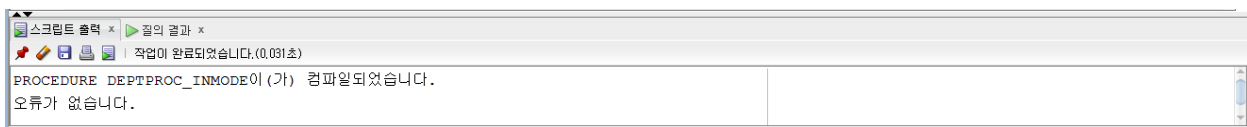
```

CREATE OR REPLACE PROCEDURE DEPTPROC_INMODE
( DEPTNO IN DEPT01.DEPTNO%TYPE,
  DNAME IN DEPT01.DNAME%TYPE,
  LOC IN DEPT01.LOC%TYPE)
IS
BEGIN
  INSERT INTO DEPT01(DEPTNO, DNAME, LOC, CREDATE)
  VALUES(DEPTNO, DNAME, LOC, SYSDATE);
  COMMIT;

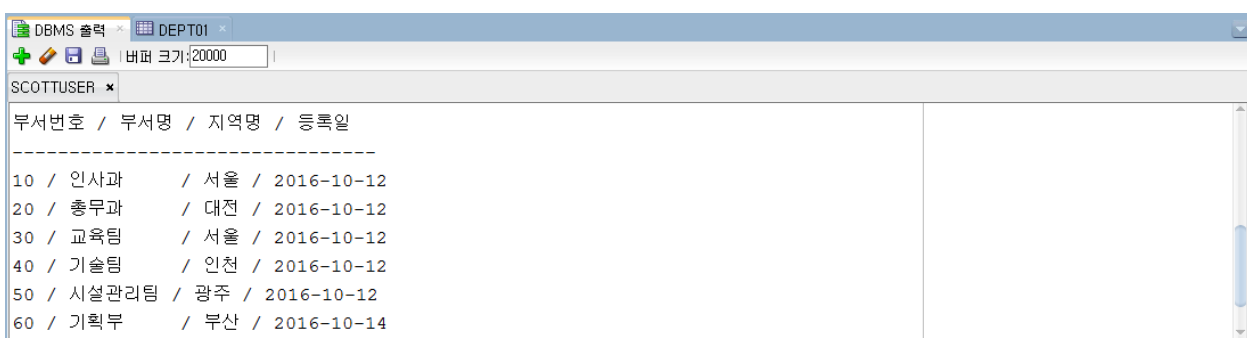
  DBMS_OUTPUT.PUT_LINE('부서번호 / 부서명 / 지역명 / 등록일');
  DBMS_OUTPUT.PUT_LINE('-----');

  FOR VDEPT IN (SELECT DEPTNO, DNAME, LOC, CREDATE FROM DEPT01 ORDER BY DEPTNO) LOOP
    DBMS_OUTPUT.PUT_LINE(VDEPT.DEPTNO || ' / ' || RPAD(VDEPT.DNAME,10) || ' / '
                        || VDEPT.LOC || ' / ' || TO_CHAR(VDEPT.CREDATE,'YYYY-MM-DD'));
  END LOOP;
END ;
/
SHOW ERROR;

```



```
EXECUTE DEPTPROC_INMODE(60, '기획부' , '부산');
```



```

CREATE OR REPLACE PROCEDURE DEPTPROC_INUP
( PDEPTNO IN DEPT01.DEPTNO%TYPE,
  PDNAME IN DEPT01.DNAME%TYPE,
  PLOC IN DEPT01.LOC%TYPE)
IS
  CNT NUMBER := 0;

```

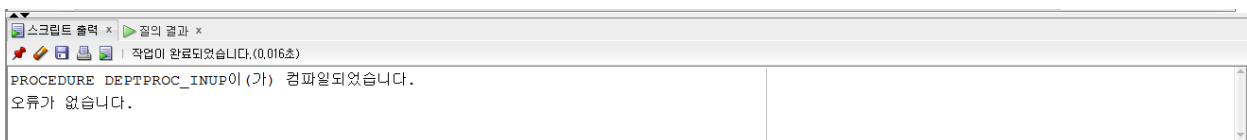


```

VDEPT DEPT01%ROWTYPE;
BEGIN
  SELECT COUNT(*) INTO CNT FROM DEPT01 WHERE DEPTNO = PDEPTNO;
  IF CNT = 0 THEN
    INSERT INTO DEPT01(DEPTNO, DNAME, LOC, CREDATE)
    VALUES(PDEPTNO, PDNAME, PLOC, SYSDATE);
  ELSE
    UPDATE DEPT01
    SET DNAME = PDNAME, LOC = PLOC, CREDATE = SYSDATE
    WHERE DEPTNO = PDEPTNO;
  END IF;
  COMMIT;

  DBMS_OUTPUT.PUT_LINE('부서번호 / 부서명 / 지역명 / 등록일');
  DBMS_OUTPUT.PUT_LINE('-----');
  SELECT DEPTNO, DNAME, LOC, CREDATE INTO VDEPT
  FROM DEPT01 WHERE DEPTNO = PDEPTNO;
  DBMS_OUTPUT.PUT_LINE(VDEPT.DEPTNO || ' / ' || RPAD(VDEPT.DNAME,10) || ' / '
                        || VDEPT.LOC || ' / ' || TO_CHAR(VDEPT.CREDATE,'YYYY-MM-DD'));
END ;
/
SHOW ERROR;

```



```
EXECUTE DEPTPROC_INUP(60, '기획부' , '전주');
```



```
EXECUTE DEPTPROC_INUP(70, '영업부' , '서울');
```



• **OUT MODE 매개변수**(프로시저 호출 후 해당 매개변수 값을 받아 사용 가능)

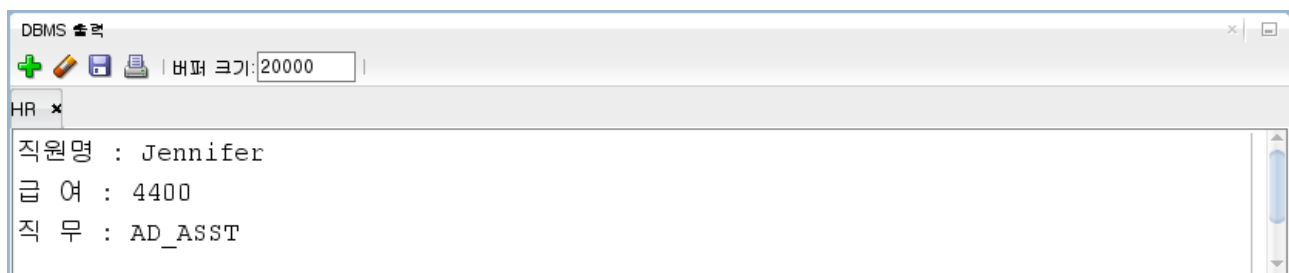
프로시저에 구한 결과 값을 얻어 내기 위해서는 MODE를 OUT으로 지정한다. OUT 매개변수는 프로시저 내에서 로직 처리 후, 해당 매개변수에 값을 할당해 프로시저 호출 부분에서 이 결과값을 참조할 수 있다.

사원 번호로 특정 고객을 조회하기 때문에 사원 번호는 IN으로 지정하고 조회해서 얻은 고객의 정보 중에서 고객의 이름과 급여와 담당 업무를 얻어오기 위해서 이름과 급여와 담당 업무컬럼을 OUT으로 지정한다.

```
CREATE OR REPLACE PROCEDURE EMPPROC_OUTMODE(  
  VEMPNO IN EMPLOYEES.EMPLOYEE_ID%TYPE,  
  VENAME OUT EMPLOYEES.FIRST_NAME%TYPE,  
  VSAL OUT EMPLOYEES.SALARY%TYPE,  
  VJOB OUT EMPLOYEES.JOB_ID%TYPE  
)  
IS  
BEGIN  
  SELECT FIRST_NAME, SALARY, JOB_ID INTO VENAME, VSAL, VJOB  
  FROM EMPLOYEES  
  WHERE EMPLOYEE_ID = VEMPNO;  
END;  
/
```

블록 내에서는 프로시저 호출은 EXECUTE 생략하고 프로시저명만 명시하면 된다.

```
DECLARE  
  VEMP EMPLOYEES%ROWTYPE;  
BEGIN  
  EMPPROC_OUTMODE(200, VEMP.FIRST_NAME, VEMP.SALARY, VEMP.JOB_ID);  
  DBMS_OUTPUT.PUT_LINE('직원명 : ' || VEMP.FIRST_NAME );  
  DBMS_OUTPUT.PUT_LINE('급여 : ' || VEMP.SALARY );  
  DBMS_OUTPUT.PUT_LINE('직무 : ' || VEMP.JOB_ID );  
END;  
/
```



모드가 OUT으로 지정한 매개변수는 프로시저에서 실행한 결과를 되돌려 받아야 하므로 반드시 매개 변수에 변수를 지정해야 한다. 이렇게 프로시저 수행 후 구해진 결과를 저장하기 위해 사용하는 변수를 **바인드 변수**라고 한다. 프로시저의 매개변수와 바인드 변수의 이름을 동일한 필요는 없지만, 자료형은 반드시 동일해야 한다. 이렇게 해서 얻어온 프로시저의 결과값을 PRINT명령을 이용해서 출력한다.

```
VARIABLE VENAME VARCHAR2(10)
```

```
VARIABLE VSAL NUMBER
```

```
VARIABLE VJOB VARCHAR2(9)
```

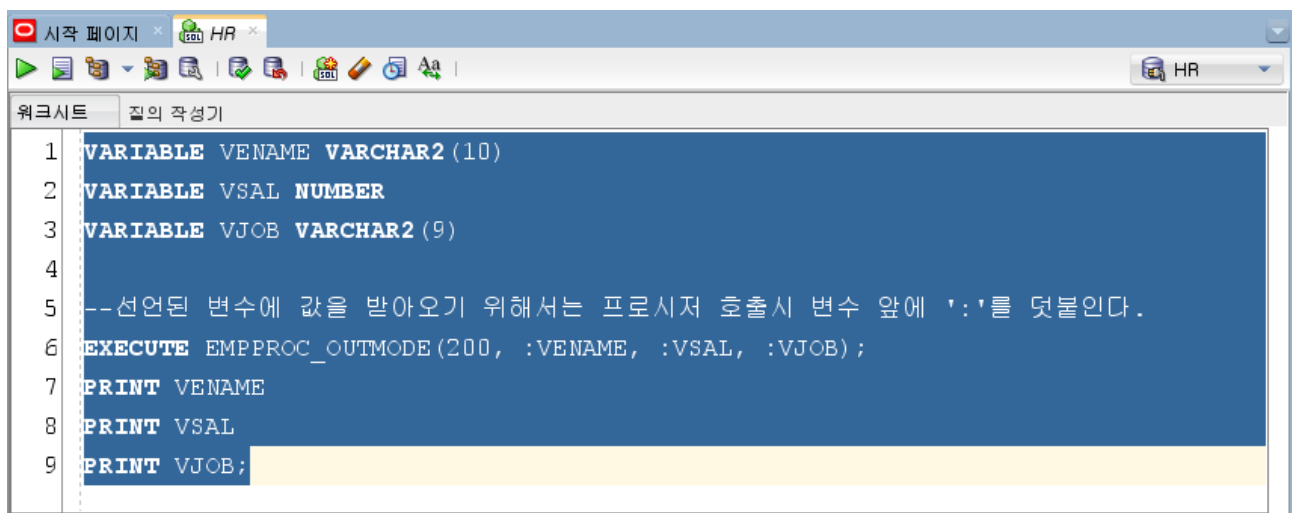
--선언된 변수에 값을 받아오기 위해서는 프로시저 호출시 변수 앞에 ':'를 덧붙인다.

```
EXECUTE EMPPROC_OUTMODE(200, :VENAME, :VSAL, :VJOB);
```

```
PRINT VENAME
```


```
PRINT VSAL
```

```
PRINT VJOB;
```



```

1 VARIABLE VENAME VARCHAR2 (10)
2 VARIABLE VSAL NUMBER
3 VARIABLE VJOB VARCHAR2 (9)
4
5 --선언된 변수에 값을 받아오기 위해서는 프로시저 호출시 변수 앞에 ':'를 덧붙인다.
6 EXECUTE EMPPROC_OUTMODE(200, :VENAME, :VSAL, :VJOB);
7 PRINT VENAME
8 PRINT VSAL
9 PRINT VJOB;
  
```



PL/SQL 프로시저가 성공적으로 완료되었습니다.
VENAME
Jennifer
VSAL
4400
VJOB
AD_ASST

• IN OUT MODE 매개변수

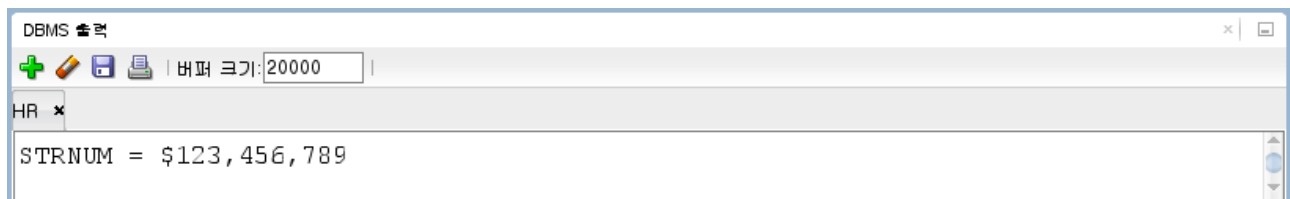
매개변수를 통해 결과값을 입력 받아 다시 해당 매개변수값으로 변형된 DATA를 받는 형태이다.

IN과 OUT의 기능을 모두 수행한다. 즉, 실행 환경에서 프로시저로 값을 전달하고 프로시저에서 실행환경으로 변경된 값을 전달할 수 있다.

숫자를 입력하면(IN), 입력한 숫자를 통화단위 형식으로 변환하여 반환(OUT)

```
CREATE OR REPLACE PROCEDURE PROC_INOUTMODE
(V_SAL IN OUT VARCHAR2)
IS
BEGIN
    V_SAL := '$' || SUBSTR(V_SAL, -9, 3) || ',' || SUBSTR(V_SAL, -6, 3) || ',' || SUBSTR(V_SAL, -3, 3);
END PROC_INOUTMODE;
/
```

```
DECLARE
    STRNUM VARCHAR2(20) := '123456789';
BEGIN
    PROC_INOUTMODE (STRNUM);
    DBMS_OUTPUT.PUT_LINE('STRNUM = ' || STRNUM);
END;
```



(2) 패키지

패키지의 사전적인 의미는 꾸러미이다.

관련 있는 프로시저를 보다 효율적으로 관리하기 위해서 패키지 단위로 배포할 때 유용하게 사용된다.

패키지는 패키지 선언(명세부)과 패키지 몸체 선언(몸체부) 두 가지 모두를 정의해야 한다.

패키지 선언부(Specification)
CREATE OR REPLACE PACKAGE 패키지명 IS TYPE_구문; 상수명 CONSTANT 상수_타입; 변수명 변수_타입; 커서구문; FUNCTION 함수명 (매개변수1 IN 매개변수1_타입, 매개변수2 IN 매개변수2_타입, ...) RETURN 반환타입; PROCEDURE 프로시저명 (매개변수1 [IN, OUT, INOUT] 매개변수1_타입, 매개변수2 [IN, OUT, INOUT] 매개변수2_타입, ...); ... END 패키지명;
패키지 본문 (Body)
CREATE OR REPLACE PACKAGE BODY 패키지명 IS 상수명 CONSTANT 상수_타입; 변수명 변수_타입; 커서정의구문; FUNCTION 함수명 (매개변수1 IN 매개변수1_타입, 매개변수2 IN 매개변수2_타입,...) RETURN 반환타입 IS BEGIN RETURN 값; END 함수명; PROCEDURE 프로시저명 (매개변수1 [IN, OUT, INOUT] 매개변수1_타입, 매개변수2 [IN, OUT, INOUT] 매개변수2_타입,...) IS BEGIN

```
....  
END 프로시저명;
```

```
...
```

```
END 패키지명;
```

```
패키지 내의 정의된 프로시저를 호출하는 방식 ["패키지명.서브_프로그램명" 형태로 사용]
```

```
EXECUTE 패키지명.프로시저명;
```

```
CREATE OR REPLACE PACKAGE EMPPACK
```

```
IS
```

```
PROCEDURE EMPPROC;
```

```
PROCEDURE EMPPROC02(VDEPTNO IN EMPLOYEES.DEPARTMENT_ID%TYPE);
```

```
END;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY EMPPACK
```

```
IS
```

```
PROCEDURE EMPPROC
```

```
IS
```

```
VWORD VARCHAR2(1);
```

```
VEMP EMPLOYEES%ROWTYPE;
```

```
CURSOR C1 (VWORD VARCHAR2)
```

```
IS
```

```
SELECT EMPLOYEE_ID, FIRST_NAME, SALARY
```

```
FROM EMPLOYEES
```

```
WHERE FIRST_NAME LIKE '%' || VWORD || '%';
```

```
BEGIN
```

```
VWORD := DBMS_RANDOM.STRING('U',1);
```

```
DBMS_OUTPUT.PUT_LINE('임의의 문자 : ' || VWORD);
```

```
OPEN C1(VWORD);
```

```
DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여');
```

```
DBMS_OUTPUT.PUT_LINE('-----');
```

```
LOOP
```

```
FETCH C1 INTO VEMP.EMPLOYEE_ID, VEMP.FIRST_NAME, VEMP.SALARY;
```

```
IF C1%ROWCOUNT = 0 THEN
```

```
DBMS_OUTPUT.PUT_LINE('해당 직원이 존재하지 않습니다') ;
```

```
END IF;
```

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

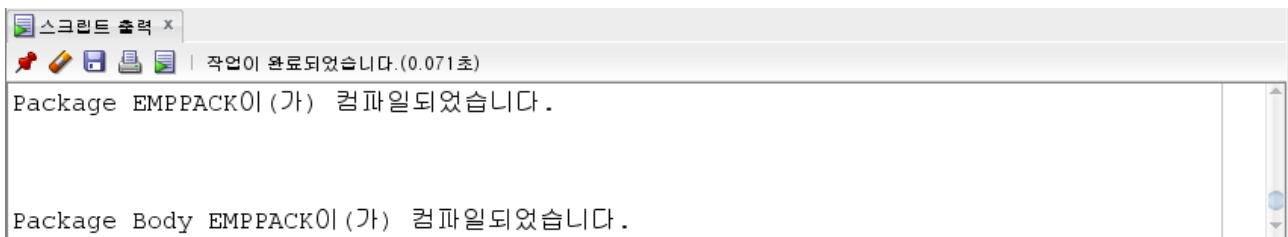
```
DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' || VEMP.SALARY);
```

```

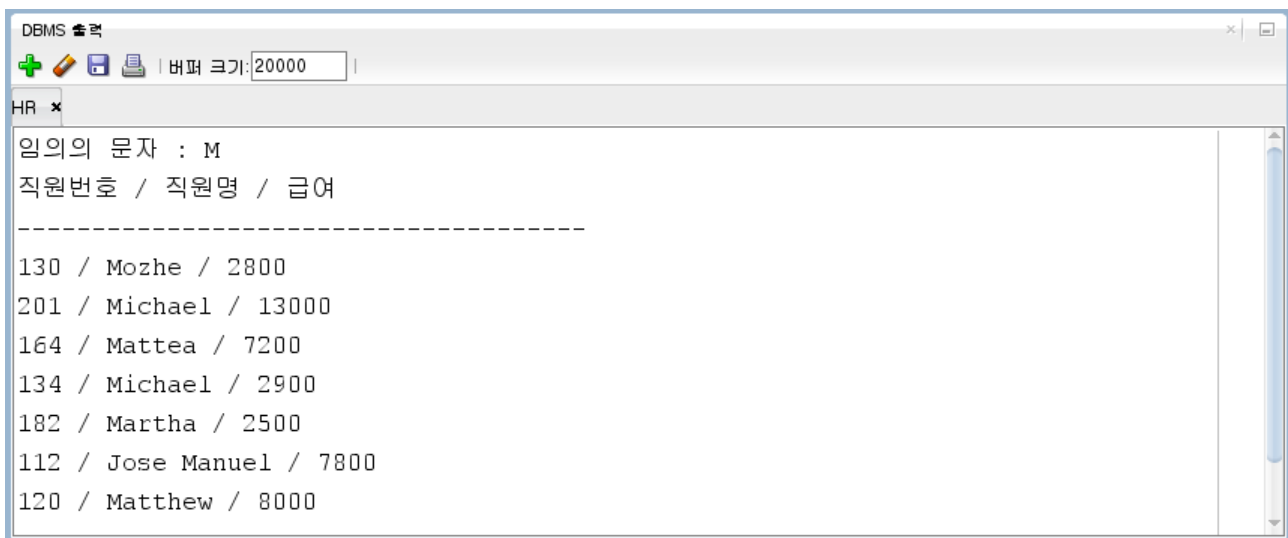
        END LOOP;
    END;

    PROCEDURE EMPPROC02( VDEPTNO IN EMPLOYEES.DEPARTMENT_ID%TYPE )
    IS
        CURSOR C1
        IS
            SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID=VDEPTNO;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('직원번호 / 직원명 / 급여');
        DBMS_OUTPUT.PUT_LINE('-----');
        FOR VEMP IN C1 LOOP
            DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' || VEMP.SALARY);
        END LOOP;
    END;
END;
/

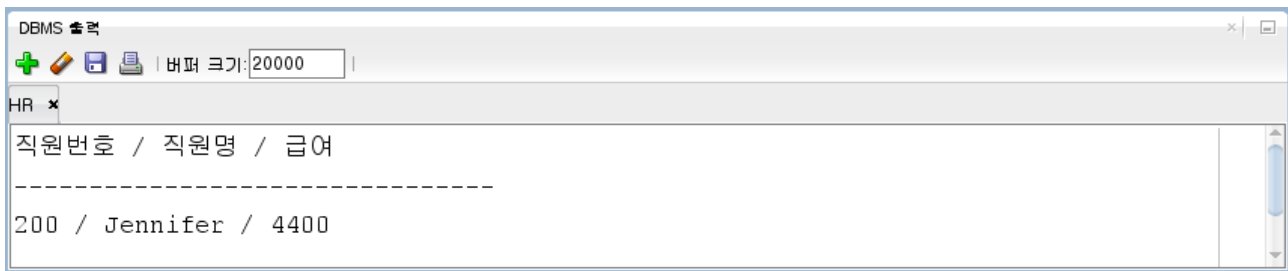
```



EXECUTE EMPPACK.EMPPROC;



EXECUTE EMPPACK.EMPPROC02(10);



```
DROP TABLE EMP02;

CREATE TABLE EMP02
AS
SELECT EMPLOYEE_ID, FIRST_NAME, HIRE_DATE
FROM EMPLOYEES;

-- 퇴사일자(RETIRE_DATE) 컬럼 추가
ALTER TABLE EMP02
ADD(RETIRE_DATE DATE);
```

```
-- 패키지 선언부
CREATE OR REPLACE PACKAGE EMP02_PKG IS
  -- 사번을 받아 이름을 반환하는 함수
  FUNCTION FN_GET_EMP02_NAME( VEMPNO IN NUMBER )
  RETURN VARCHAR2;

  -- 신규 직원 입력
  PROCEDURE NEW_EMP02_PROC
  (VENAME IN EMP02.FIRST_NAME%TYPE, VHIREDATE EMP02.HIRE_DATE%TYPE);

  -- 퇴사 직원 처리
  PROCEDURE RETIRE_EMP02_PROC(VEMPNO IN EMP02.EMPLOYEE_ID%TYPE);
END EMP02_PKG;
/
```

```
-- 패키지 본문
CREATE OR REPLACE PACKAGE BODY EMP02_PKG IS

  -- 직원번호를 받아 이름을 반환하는 함수
  FUNCTION FN_GET_EMP02_NAME( VEMPNO IN NUMBER )
  RETURN VARCHAR2
  IS
```



```

VENAME EMP02.FIRST_NAME%TYPE;
BEGIN
  -- 직원명을 가져온다.
  SELECT FIRST_NAME
    INTO VENAME
    FROM EMPLOYEES
   WHERE EMPLOYEE_ID = VEMPNO;
  -- 직원명 반환
  RETURN NVL(VENAME, '해당 직원 없음');
END FN_GET_EMP02_NAME;

```

```

-- 신규 직원 입력
PROCEDURE NEW_EMP02_PROC
(VENAME IN EMP02.FIRST_NAME%TYPE, VHIREDATE IN EMP02.HIRE_DATE%TYPE)
IS
  VEMPNO EMP02.EMPLOYEE_ID%TYPE;
BEGIN
  -- 신규사원의 사번 = 최대 사번+1
  SELECT NVL(MAX(EMPLOYEE_ID),0) + 1
    INTO VEMPNO
    FROM EMPLOYEES;

  INSERT INTO EMP02(EMPLOYEE_ID, FIRST_NAME, HIRE_DATE)
    VALUES(VEMPNO, VENAME, NVL(VHIREDATE, SYSDATE));
  COMMIT;

  EXCEPTION WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    ROLLBACK;
END NEW_EMP02_PROC;

```

```

-- 퇴사 직원 처리
PROCEDURE RETIRE_EMP02_PROC(VEMPNO IN EMP02.EMPLOYEE_ID%TYPE)
IS
  CNT NUMBER := 0;
  E_NO_DATA EXCEPTION;
BEGIN
  -- 퇴사한 직원은 직원테이블에서 삭제하지 않고 일단 퇴사일자(RETIRE_DATE)를 NULL에서 갱신
  한다.
  UPDATE EMP02 SET RETIRE_DATE = SYSDATE

```

```
WHERE EMPLOYEE_ID = VEMPNO AND RETIRE_DATE IS NULL;
```

```
-- UPDATE된 건수를 가져온다.
```

```
CNT := SQL%ROWCOUNT;
```

```
-- 갱신된 건수가 없으면 사용자 예외처리
```

```
IF CNT = 0 THEN
```

```
    RAISE E_NO_DATA;
```

```
END IF;
```

```
COMMIT;
```

```
EXCEPTION
```

```
    WHEN E_NO_DATA THEN
```

```
        DBMS_OUTPUT.PUT_LINE (VEMPNO || '에 해당되는 퇴사처리할 직원이 없습니다!');
```

```
        ROLLBACK;
```

```
    WHEN OTHERS THEN
```

```
        DBMS_OUTPUT.PUT_LINE (SQLERRM);
```

```
        ROLLBACK;
```

```
END RETIRE_EMP02_PROC;
```

```
END EMP02_PKG;
```

```
/
```

```
SELECT EMP02_PKG.FN_GET_EMP02_NAME(200) FROM DUAL;
```

EMP02_PKG.FN_GET_EMP02_NAME(200)
1 Jennifer

```
EXECUTE EMP02_PKG.NEW_EMP02_PROC('Roberts', '2021-01-01');
```

```
SELECT *
```

```
FROM EMP02
```

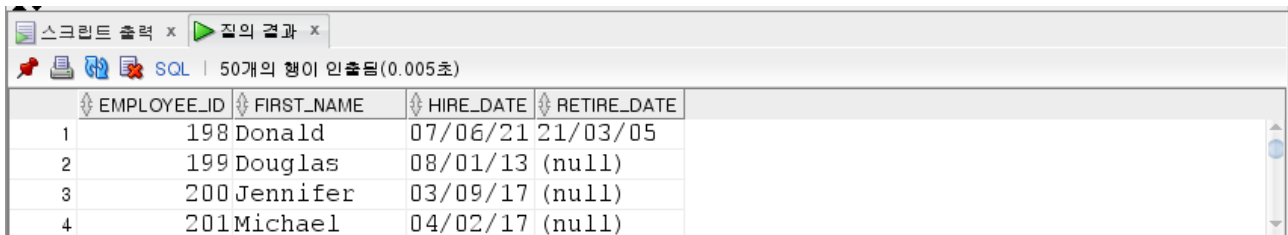
```
WHERE FIRST_NAME = 'Roberts';
```

EMPLOYEE_ID	FIRST_NAME	HIRE_DATE	RETIRE_DATE
1	207 Roberts	21/01/01	(null)

퇴사 직원

```
EXECUTE EMP02_PKG.RETIRE_EMP02_PROC(198);
```

```
SELECT *  
FROM EMP02;
```



	EMPLOYEE_ID	FIRST_NAME	HIRE_DATE	RETIRE_DATE
1	198	Donald	07/06/21	21/03/05
2	199	Douglas	08/01/13	(null)
3	200	Jennifer	03/09/17	(null)
4	201	Michael	04/02/17	(null)

DBMS_OUTPUT 패키지

오라클을 설치하게 되면 특정 목적을 위해서 사용할 수 있도록 오라클사에서 제공해 주는 패키지들이 설치된다. 이러한 패키지에는 오라클에서 제공되는 프로시저와 함수들을 관련된 것끼리 묶어서 집합으로 제공한다. 패키지에 대한 정보를 출력해 보도록 하자.

```
DESC DBA_OBJECTS;
```

패키지들은 목적에 따라 관련된 프로시저와 함수들을 관리한다.

```
SELECT OBJECT_NAME FROM DBA_OBJECTS  
WHERE OBJECT_TYPE='PACKAGE' AND OBJECT_NAME LIKE 'DBMS_%'  
ORDER BY OBJECT_NAME;
```

(3) 트리거

트리거(trigger)의 사전적인 의미는 방아쇠나 (방아쇠를) 쏘다, 발사하다, (사건을) 유발시키다라는 의미가 있다. PL/SQL에서의 트리거 역시 방아쇠가 당겨지면 자동으로 총알이 발사되듯이 어떠한 이벤트가 발생하면 그에 따라 다른 작업이 자동으로 처리되는 것을 의미한다.

트리거란 특정테이블의 데이터에 변경이 가해졌을 때 자동으로 수행되는 저장 프로시저라고 할 수 있다. 앞서 배운 저장 프로시저는 필요할 때마다 사용자가 직접 EXECUTE 명령어로 호출해야 했다. 하지만 트리거는 이와 달리 테이블의 데이터가 INSERT, UPDATE, DELETE 문에 의해 변경 될때 자동으로 수행되므로 이 기능을 이용하며 여러 가지 작업을 할 수 있다. 이런 이유로 트리거를 사용자가 직접 실행시킬 수는 없다.

```
CREATE TRIGGER trigger_name
Timing[BEFORE|AFTER]          --- ①
EVENT[INSERT|UPDATE|DELETE]  --- ②
ON table_name
[FOR EACH ROW]
[WHEN conditions]
BEGIN
    statement
END
```

① 트리거의 타이밍(트리거의 동작 시점을 설정)

[BEFORE] 타이밍은 어떤 테이블에 INSERT, UPDATE, DELETE 문이 실행될 때 해당 문장이 실행되기 전에 트리거가 가지고 있는 BEGIN ~ END 사이의 문장을 실행한다.

[AFTER] 타이밍은 INSERT, UPDATE, DELETE 문이 실행되고 난 후에 트리거가 가지고 있는 BEGIN ~ END 사이의 문장을 실행한다.

② 트리거의 이벤트(트리거 동작을 위한 이벤트 종류를 정의하는 단계)

트리거의 이벤트는 사용자가 어떤 DML(INSERT, UPDATE, DELETE)문을 실행했을 때 트리거를 발생시킬 것인지를 결정한다. ON 테이블명은 트리거가 주목하는 대상 테이블을 정의한다.

③ 트리거의 유형

트리거의 유형은 FOR EACH ROW에 의해 문장 레벨 트리거와 행 레벨 트리거로 나눈다.

FOR EACH ROW가 생략되면 문장 레벨 트리거이고, 행 레벨 트리거를 정의하고자 할 때에는 반드시 FOR EACH ROW를 기술해야만 한다. 이때 행 레벨 트리거는 추가되는 행의 수만큼 트리거가 동작하여 행 내에서 이벤트 발생되는걸 기준으로 트리거의 감시, 보완 범위를 정해두는 내용이다.

문장 레벨 트리거는 어떤 사용자가 트리거가 설정되어 있는 테이블에 대해 DML(INSERT, UPDATE, DELETE)문을 실행할 때 단 한번만 트리거를 발생시킬 때 사용한다.

행 레벨 트리거는 DML(INSERT, UPDATE, DELETE)문에 의해서 여러 개의 행이 변경된다면 각 행이 변경 될 때마다 트리거를 발생시키는 방법이다. 만약 5개의 행이 변경되면 5번 트리거가 발생된다.

④ :NEW & :OLD

행 레벨 트리거에서 컬럼의 실제 데이터 값을 제어하는데 사용되는 연산자로 INSERT문의 경우 입력된 컬럼의 값은 :NEW, DELETE문의 경우 삭제되는 컬럼값은 :OLD, UPDATE문의 경우 변경 전 컬럼 데이터 값은 :OLD로 수정할 새로운 데이터값은 :NEW로 가진다.

⑤ 트리거의 몸체(BEGIN ~ END)

트리거의 몸체는 해당 타이밍에 해당 이벤트가 발생하게 되면 실행될 기본 로직이 포함되는 부분으로 BEGIN ~ END에 기술한다.

- 트리거의 사용목적

- 데이터베이스 테이블 생성하는 과정에서 참조 무결성과 데이터 무결성 등의 복잡한 제약 조건을 생성하는 경우
- 데이터베이스 테이블의 데이터에 생기는 작업을 감시, 보완
- 데이터베이스 테이블에 생기는 변화에 따라 필요한 다른 프로그램을 실행하는 경우
- 불필요한 트랜잭션을 금지하기 위해
- 컬럼의 값을 자동으로 생성되도록 하는 경우

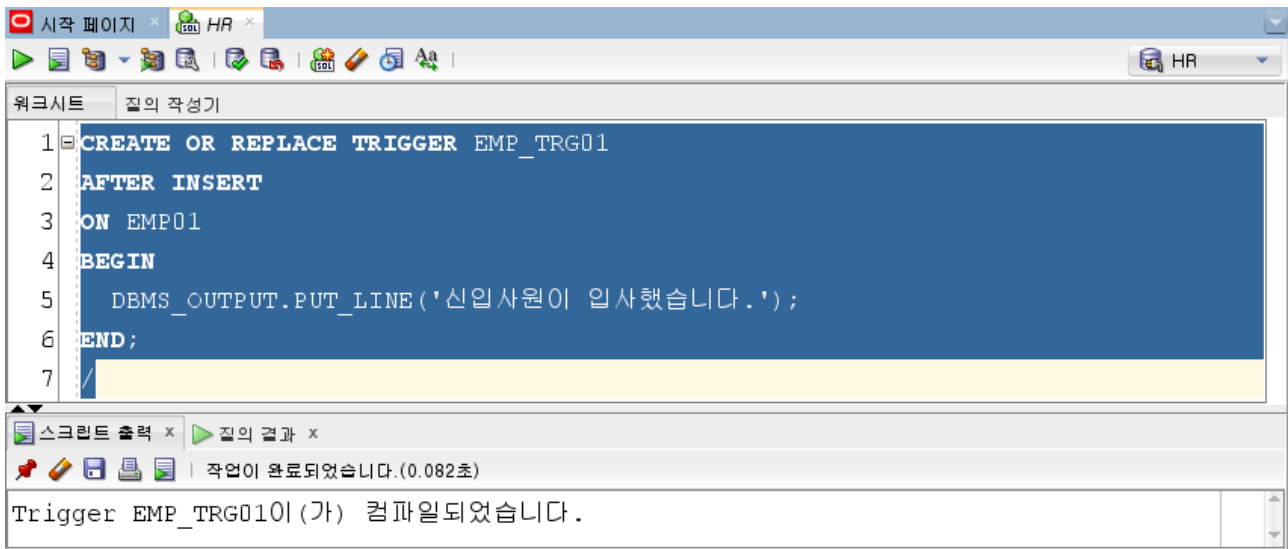
```
DROP TABLE EMP01;
```

```
CREATE TABLE EMP01(  
    EMPNO NUMBER(4) PRIMARY KEY,  
    ENAME VARCHAR2(20),  
    JOB VARCHAR2(50)  
);
```

-- 사원 테이블에 로우가 추가되면 자동 수행할 트리거를 생성한다.

```
CREATE OR REPLACE TRIGGER EMP_TRG01  
AFTER INSERT  
ON EMP01  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('신입사원이 입사했습니다.');
```

```
END;  
/
```

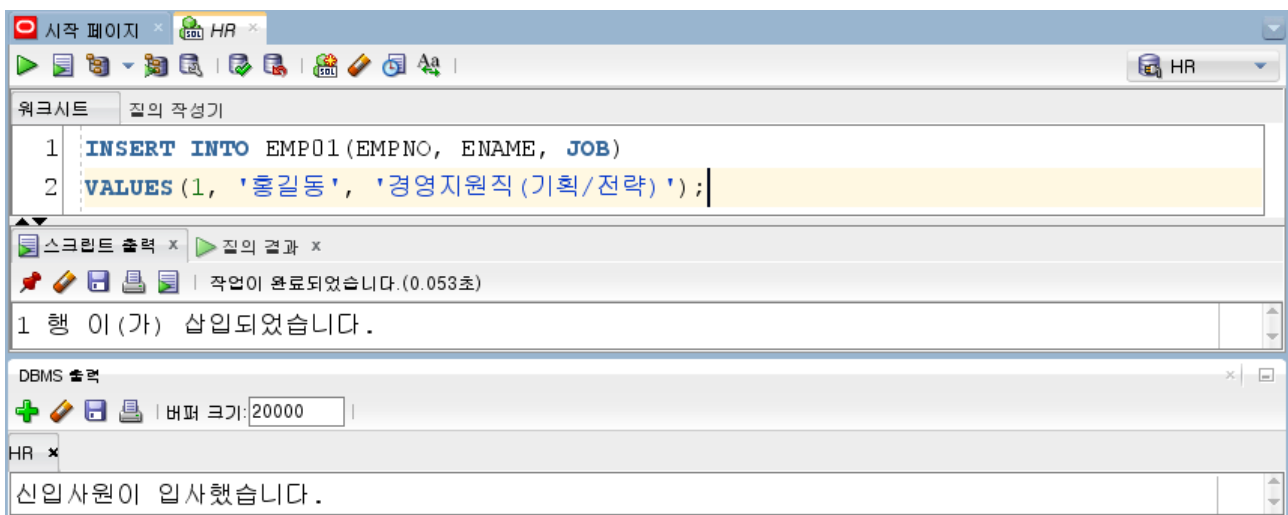


-- 사원 테이블에 로우를 추가한다.

```

INSERT INTO EMP01(EMPNO, ENAME, JOB)
VALUES(1, '홍길동', '경영지원직(기획/전략)');

```

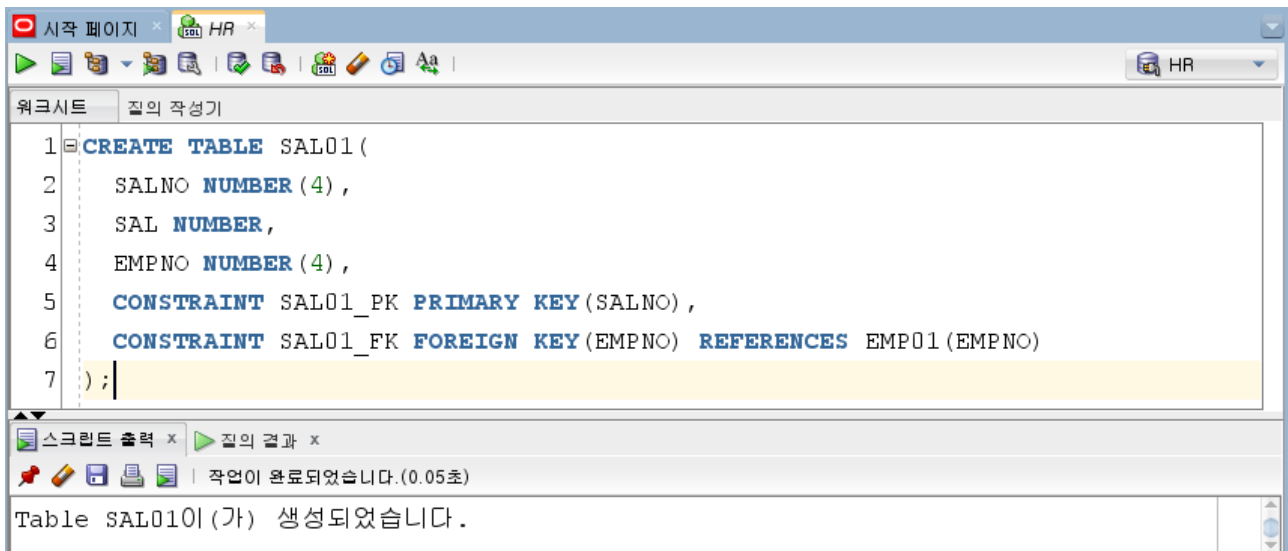


-- 사원 테이블에 새로운 데이터 즉 신입 사원이 들어오면 급여 테이블에 새로운 데이터를 자동으로 생성하고 싶을 경우, 사원 테이블에 트리거를 설정하여 구현할 수 있다.

```

CREATE TABLE SAL01(
    SALNO NUMBER(4),
    SAL NUMBER,
    EMPNO NUMBER(4),
    CONSTRAINT SAL01_PK PRIMARY KEY(SALNO),
    CONSTRAINT SAL01_FK FOREIGN KEY(EMPNO) REFERENCES EMP01(EMPNO)
);

```

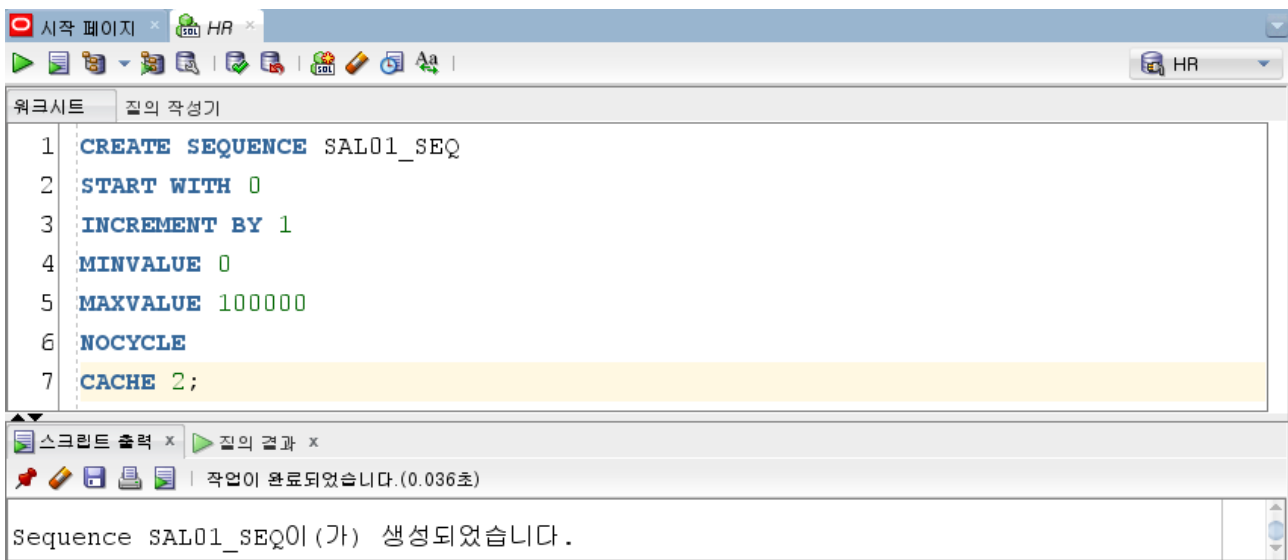


-- 급여 번호가 PRIMARY KEY이므로 중복된 데이터를 저장할 수 없다. 그래서 급여번호를 자동 생성하는 시퀀스를 정의하고 이 시퀀스로부터 일련번호를 얻어 급여번호에 부여한다.

```

CREATE SEQUENCE SAL01_SEQ
START WITH 0
INCREMENT BY 1
MINVALUE 0
MAXVALUE 100000
NOCYCLE
CACHE 2;

```



-- 트리거 생성

```

CREATE OR REPLACE TRIGGER EMP_TRG02
AFTER INSERT
ON EMP01
FOR EACH ROW

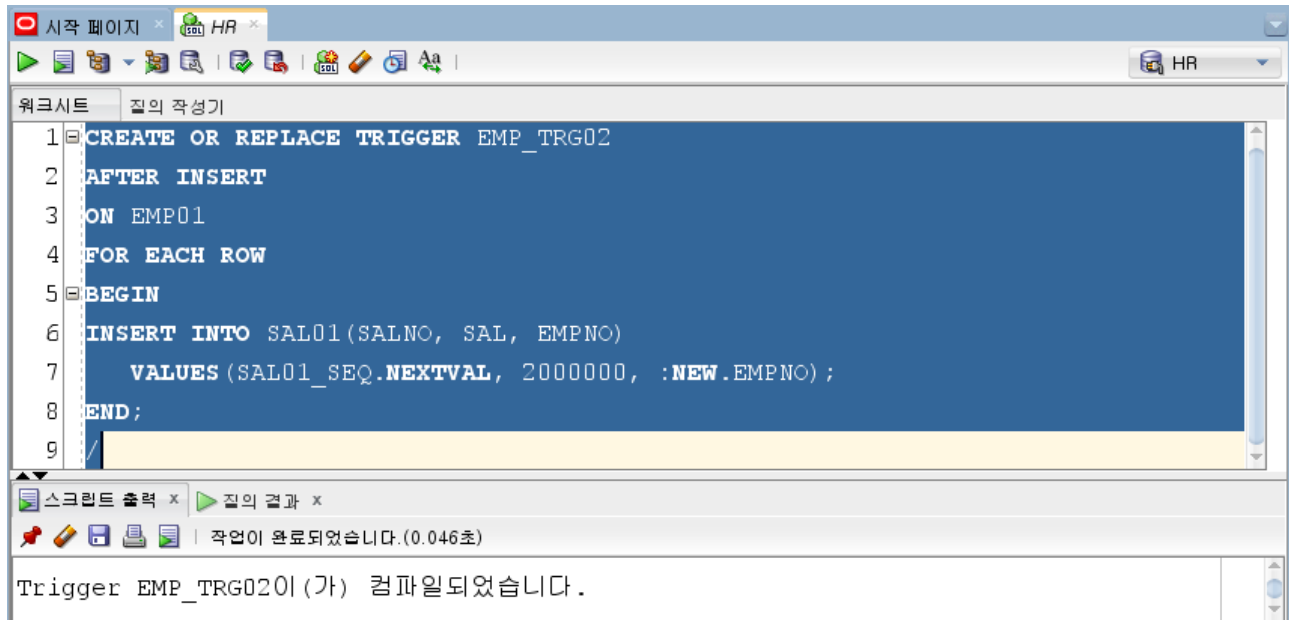
```

```

BEGIN
    INSERT INTO SAL01(SALNO, SAL, EMPNO)
    VALUES(SAL01_SEQ.NEXTVAL, 2000000, :NEW.EMPNO);
END;
/

```

실행방법은 위 트리거 소스를 블록 설정하고 마우스 오른쪽 클릭 → 스크립트 실행(F5) 클릭하면 된다.

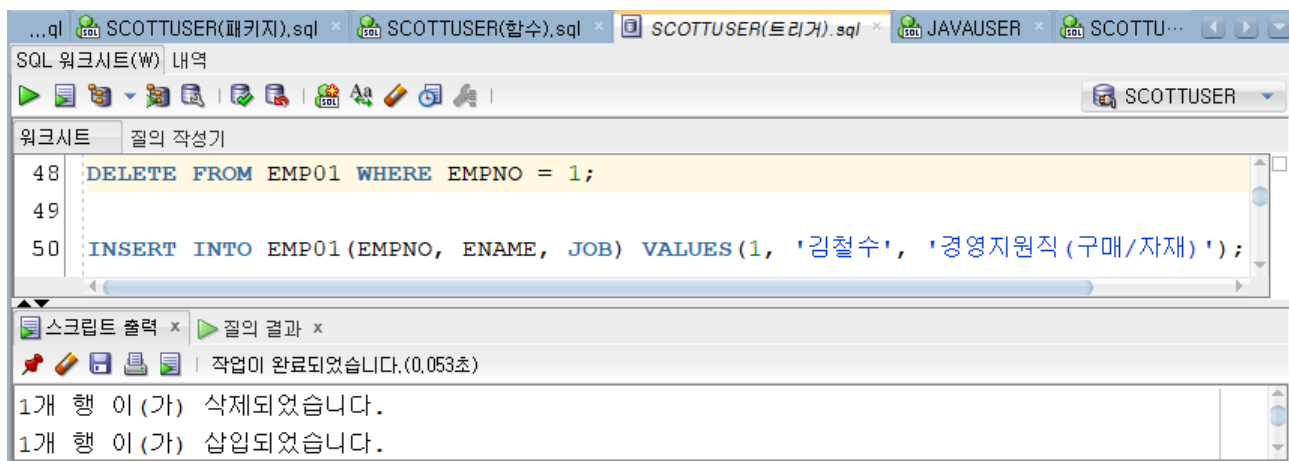


-- 사원 테이블에 로우를 추가한다.

```

DELETE FROM EMP01 WHERE EMPNO = 1;
INSERT INTO EMP01(EMPNO, ENAME, JOB) VALUES(1, '김철수', '경영지원직(구매/자재)');

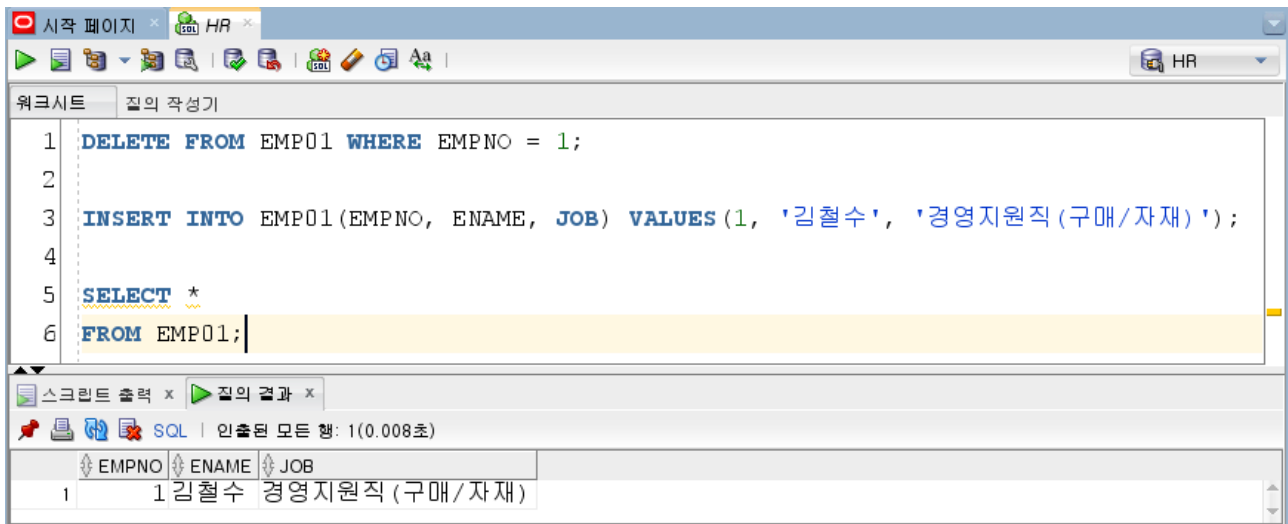
```



```

SELECT *
FROM EMP01;

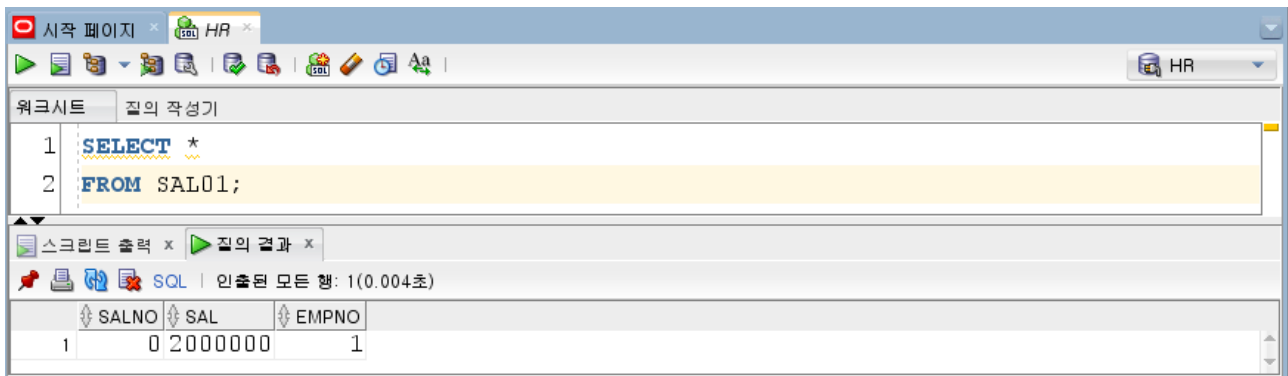
```

```

SELECT *
FROM SAL01;

```



```

INSERT INTO EMP01(EMPNO, ENAME, JOB) VALUES(2, '이영희', '경영지원직(인사)');

```

```

SELECT * FROM EMP01;

```

```

SELECT * FROM SAL01;

```

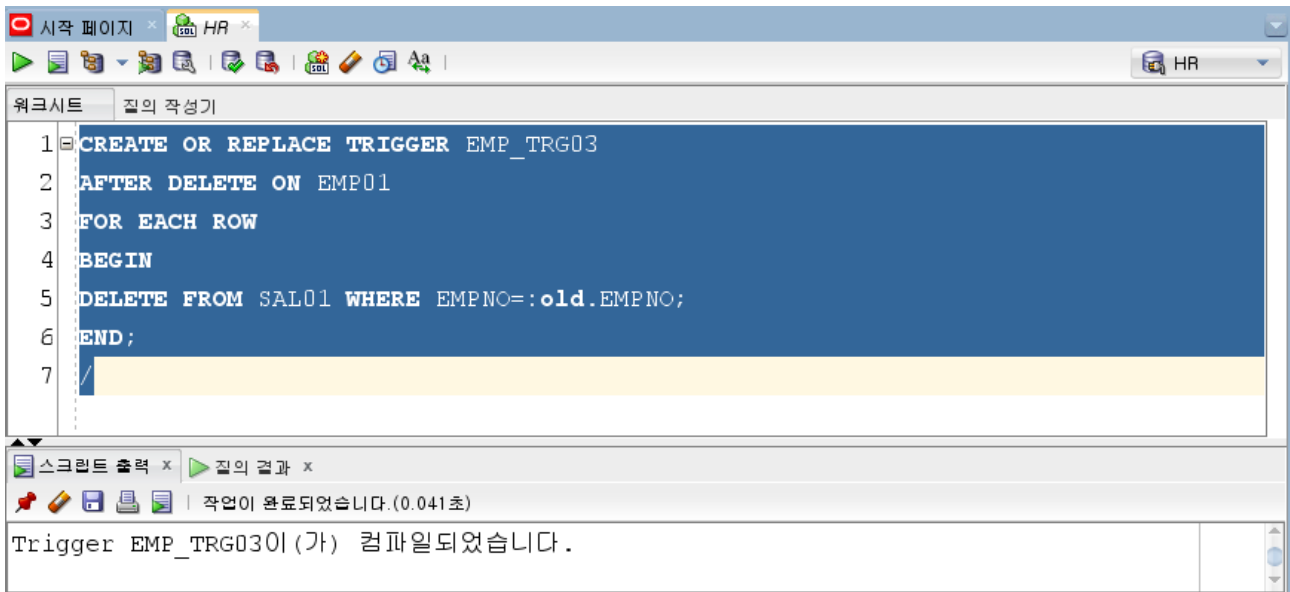
EMPNO	ENAME	JOB	SALNO	SAL	EMPNO
1	1 김철수	경영지원직 (구매/자재)	1	0 2000000	1
2	2 이영희	경영지원직 (인사)	2	1 2000000	2

-- 사원의 정보가 제거될 때 그 사원의 급여 정보도 함께 삭제하는 내용을 트리거로 작성한다.

```

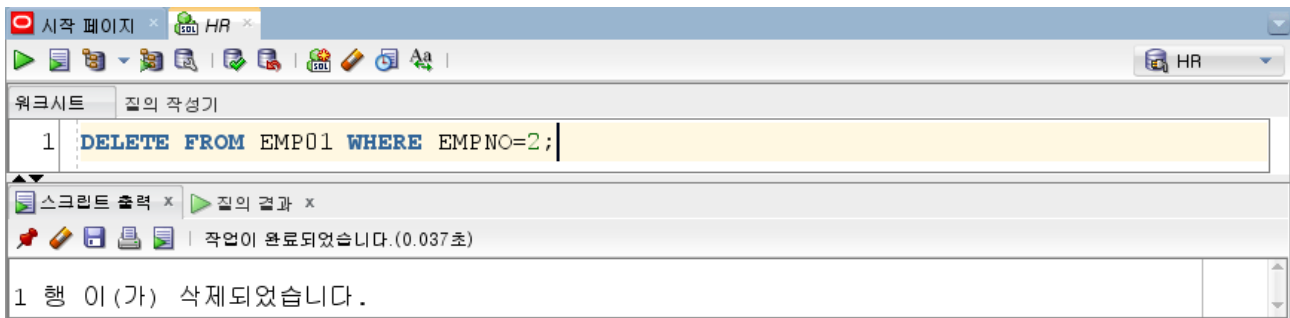
CREATE OR REPLACE TRIGGER EMP_TRG03
AFTER DELETE ON EMP01
FOR EACH ROW
BEGIN
    DELETE FROM SAL01 WHERE EMPNO=:old.EMPNO;
END;
/

```



-- 사원테이블의 로우를 삭제한다

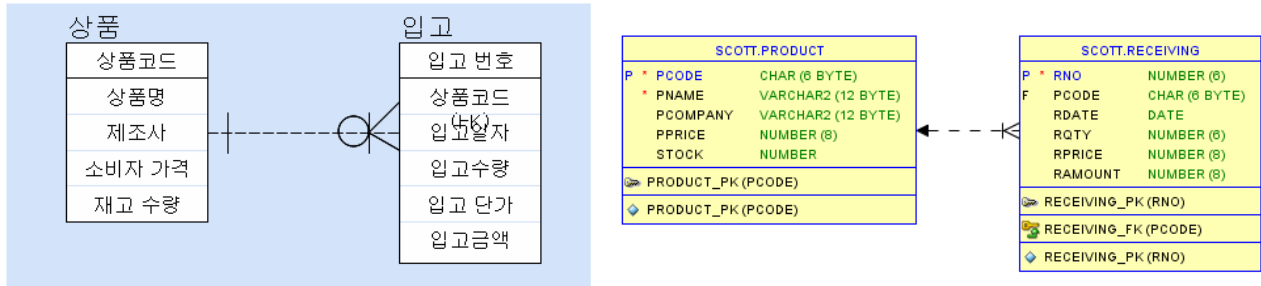
```
DELETE FROM EMP01 WHERE EMPNO=2;
```



SELECT * FROM EMP01;	SELECT * FROM SAL01;
----------------------	----------------------

EMPNO	ENAME	JOB	SALNO	SAL	EMPNO
1	1 김철수	경영지원직 (구매/자재)	1	0 2000000	1

• 예제를 통한 트리거의 적용



<실습하기> 입고 트리거 작성하기

입고 테이블에 상품이 입력되면 입고 수량을 상품 테이블의 재고 수량에 추가하는 트리거 작성해 보자.

1. 상품 테이블을 생성.

```
CREATE TABLE PRODUCT(
    PCODE CHAR(6),                -- 상품코드
    PNAME VARCHAR2(12) NOT NULL,  -- 상품명
    PCOMPANY VARCHAR(12),         -- 제조사
    PPRICE NUMBER(8),             -- 가격
    STOCK NUMBER DEFAULT 0,       -- 재고수량
    CONSTRAINT PRODUCT_PK PRIMARY KEY(PCODE)
);
```



2. 입고 테이블을 생성

```
CREATE TABLE RECEIVING(
    RNO NUMBER(6),                -- 입고번호
    PCODE CHAR(6) ,              -- 상품코드
    RDATE DATE DEFAULT SYSDATE,   -- 입고날짜
    RQTY NUMBER(6),              -- 입고수량
    RPRICE NUMBER(8),            -- 입고가격
    RAMOUNT NUMBER(8),           -- 입고단가
    CONSTRAINT RECEIVING_PK PRIMARY KEY(RNO),
    CONSTRAINT RECEIVING_FK FOREIGN KEY(PCODE) REFERENCES PRODUCT(PCODE)
);
```



3. 상품테이블의 재고수량 컬럼을 통해서 실질적인 트리거의 적용 예를 살펴보도록 하겠다.
우선 상품 테이블에 다음과 같은 샘플 데이터를 입력해 보자.

```
INSERT INTO PRODUCT(PCODE, PNAME, PCOMPANY, PPRICE)
VALUES('A00001','세탁기', 'LG', 1500000);
INSERT INTO PRODUCT(PCODE, PNAME, PCOMPANY, PPRICE)
VALUES('A00002','컴퓨터', 'LG', 1000000);
INSERT INTO PRODUCT(PCODE, PNAME, PCOMPANY, PPRICE)
VALUES('A00003','냉장고', '삼성', 4500000);

SELECT * FROM PRODUCT;
```

	PCODE	PNAME	PCOMPANY	PPRICE	STOCK
1	A00001	세탁기	LG	1500000	0
2	A00002	컴퓨터	LG	1000000	0
3	A00003	냉장고	삼성	4500000	0

4. 입고 테이블에 상품이 입력되면 입고 수량을 상품 테이블의 재고 수량에 추가하는 트리거 작성.

```
CREATE OR REPLACE TRIGGER TRG_IN
AFTER INSERT ON RECEIVING
FOR EACH ROW
BEGIN
    UPDATE PRODUCT
    SET STOCK = STOCK + :NEW.RQTY --재고수량 = 재고수량 + 입고수량
    WHERE PCODE = :NEW.PCODE;
END;
/
```

5. 트리거를 실행시킨 후 입고 테이블에 행을 추가한다. 입고 테이블에는 물론 상품 테이블의 재고 수량이 변경됨을 확인할 수 있다.

```
INSERT INTO RECEIVING(RNO, PCODE, RQTY, RPRICE, RAMOUNT)
VALUES(1, 'A00001', 5, 850000, 950000);
```

```
SELECT * FROM RECEIVING;
```

	RNO	PCODE	RDATE	RQTY	RPRICE	RAMOUNT
1	1	A00001	16/10/19	5	850000	950000

```
SELECT * FROM PRODUCT;
```

	PCODE	PNAME	PCOMPANY	PPRICE	STOCK
1	A00001	세탁기	LG	1500000	5
2	A00002	컴퓨터	LG	1000000	0
3	A00003	냉장고	삼성	4500000	0

6. 입고 테이블에 상품이 입력되면 자동으로 상품 테이블의 재고 수량이 증가하게 된다. 입고 테이블에 또 다른 상품을 입력한다.

```
INSERT INTO RECEIVING(RNO, PCODE, RQTY, RPRICE, RAMOUNT)
VALUES(2, 'A00002', 10, 680000, 780000);
```

```
SELECT * FROM RECEIVING;
```

	RNO	PCODE	RDATE	RQTY	RPRICE	RAMOUNT
1	1	A00001	16/10/19	5	850000	950000
2	2	A00002	16/10/19	10	680000	780000

```
SELECT * FROM PRODUCT;
```

	PCODE	PNAME	PCOMPANY	PPRICE	STOCK
1	A00001	세탁기	LG	1500000	5
2	A00002	컴퓨터	LG	1000000	10
3	A00003	냉장고	삼성	4500000	0

```
INSERT INTO RECEIVING(RNO, PCODE, RQTY, RPRICE, RAMOUNT)
VALUES(3, 'A00003', 10, 250000, 300000);
```

```
SELECT * FROM RECEIVING;
```

	RNO	PCODE	RDATE	RQTY	RPRICE	RAMOUNT
1	1	A00001	16/10/19	5	850000	950000
2	2	A00002	16/10/19	10	680000	780000
3	3	A00003	16/10/19	10	250000	300000

```
SELECT * FROM PRODUCT;
```

	PCODE	PNAME	PCOMPANY	PPRICE	STOCK
1	A00001	세탁기	LG	1500000	5
2	A00002	컴퓨터	LG	1000000	10
3	A00003	냉장고	삼성	4500000	10

<실습하기> 갱신 트리거 작성하기

이미 입고된 상품에 대해서 입고 수량이 변경되면 상품 테이블의 재고수량 역시 변경되어야 한다. 이를 위한 갱신 트리거 작성해 보자.

1. 갱신 트리거 생성

```
CREATE OR REPLACE TRIGGER TRG_UP
AFTER UPDATE ON RECEIVING
FOR EACH ROW
BEGIN
    UPDATE PRODUCT
    SET STOCK = STOCK + (:old.RQTY+ :new.RQTY)
    WHERE PCODE = :new.PCODE;
END;
/
```

2. 입고 번호 3번은 냉장고가 입고된 정보를 기록한 것으로서 입고 번호 3번의 입고수량을 8로 변경하였더니 냉장고의 재고 수량 역시 8로 변경되었다.

```
UPDATE RECEIVING SET RQTY=8, RAMOUNT=280000 -- 입고수량과 입고금액
WHERE RNO=3;
```

```
SELECT * FROM RECEIVING;
```

	RNO	PCODE	RDATE	RQTY	RPRICE	RAMOUNT
1	1	A00001	16/10/19	5	850000	950000
2	2	A00002	16/10/19	10	680000	780000
3	3	A00003	16/10/19	8	250000	280000

```
SELECT * FROM PRODUCT;
```

	PCODE	PNAME	PCOMPANY	PPRICE	STOCK
1	A00001	세탁기	LG	1500000	5
2	A00002	컴퓨터	LG	1000000	10
3	A00003	냉장고	삼성	4500000	8

<실습하기> 삭제 트리거 작성하기

입고 테이블에서 입고되었던 상황이 삭제되면 상품 테이블에 재고수량에서 삭제된 입고수량 만큼을 빼는 삭제 트리거 작성해 보자.

1. 삭제 트리거 생성

```
CREATE OR REPLACE TRIGGER TRG_DEL
AFTER DELETE ON RECEIVING
FOR EACH ROW
BEGIN
```

```

UPDATE PRODUCT
SET STOCK = STOCK - :old.RQTY
WHERE PCODE = :old.PCODE;

END;
/

```

2. 입고 번호 3번은 냉장고가 입고된 정보를 기록한 것으로서 입고 번호가 3번인 행을 삭제하였더니 냉장고의 재고 수량 역시 0으로 변경되었다.

```
DELETE RECEIVING WHERE RNO = 3;
```

```
SELECT * FROM RECEIVING;
```

	RNO	PCODE	RDATE	RQTY	RPRICE	RAMOUNT
1	1	A00001	16/10/19	5	850000	950000
2	2	A00002	16/10/19	10	680000	780000

```
SELECT * FROM PRODUCT;
```

	PCODE	PNAME	PCOMPANY	PPRICE	STOCK
1	A00001	세탁기	LG	1500000	5
2	A00002	컴퓨터	LG	1000000	10
3	A00003	냉장고	삼성	4500000	0

(4) FUNCTION 이란?

값을 반환하는 명명된 PL/SQL BLOCK으로 오라클 내장 함수와 같이 SQL 표현식의 일부로 사용하여 복잡한 SQL문을 간단한 형태로 사용할 수 있다. 값을 반환하는 RETURN이 반드시 포함되며 반드시 하나의 값을 반환한다. 명명된 이름으로 반복적인 호출이 가능하며 중앙 집중적으로 유지/관리가 편리하다. RETURN의 데이터타입은 오직 CHAR, DATE, NUMBER여야 한다.

```
CREATE [OR REPLACE] FUNCTION FUNCTION명
(매개변수1 데이터타입, 매개변수2 데이터타입 ...)
RETURN 데이터타입 -- 함수가 반환할 데이터 타입 지정. 크기는 지정할 수 없다.
IS
지역변수
BEGIN
실행부
RETURN 반환값; -- 매개변수를 받아 특정 연산을 수행한 후 반환할 값을 명시
[EXCEPTION 예외 처리부]
END [함수이름];
/
```

-- 부서 번호를 부서 이름을 반환하는 함수를 생성하자. (첫번째 방법)

```
CREATE OR REPLACE FUNCTION GETDNAME(V_DEPTNO IN EMPLOYEES.DEPARTMENT_ID%TYPE)
RETURN VARCHAR2
IS
V_DNAME VARCHAR2(50);
V_CNT NUMBER := 0;
BEGIN
SELECT COUNT(*) INTO V_CNT FROM DEPARTMENTS
WHERE DEPARTMENT_ID = V_DEPTNO;

IF V_CNT = 0 THEN
V_DNAME := '해당 부서 없음';
ELSE
SELECT DEPARTMENT_NAME INTO V_DNAME FROM DEPARTMENTS
WHERE DEPARTMENT_ID = V_DEPTNO;
END IF;
RETURN V_DNAME;
END;
/
```

```
SELECT FIRST_NAME, JOB_ID, NVL(COMMISSION_PCT,0), SALARY, GETDNAME(DEPARTMENT_ID)
DEPARTMENT_NAME
```



```
FROM EMPLOYEES
WHERE FIRST_NAME='Steven';
```

	FIRST_NAME	JOB_ID	NVL(COMMISSION_PCT,0)	SALARY	DEPARTMENT_NAME
1	Steven	AD_PRES	0	24000	Executive
2	Steven	ST_CLERK	0	2200	Shipping

```
SELECT GETDNAME(178) FROM DUAL;
```

	GETDNAME(178)
1	해당 부서 없음

-- 부서 번호를 부서 이름을 반환하는 함수를 생성하자. (두번째 방법)

```
CREATE OR REPLACE FUNCTION GETDNAME(V_DEPTNO IN EMPLOYEES.DEPARTMENT_ID%TYPE)
RETURN VARCHAR2
IS
    V_DNAME VARCHAR2(50);
BEGIN
    SELECT DEPARTMENT_NAME INTO V_DNAME
    FROM DEPARTMENTS
    WHERE DEPARTMENT_ID = V_DEPTNO;
    RETURN V_DNAME;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        V_DNAME := '해당 부서 없음';
        RETURN V_DNAME;
END;
/
```

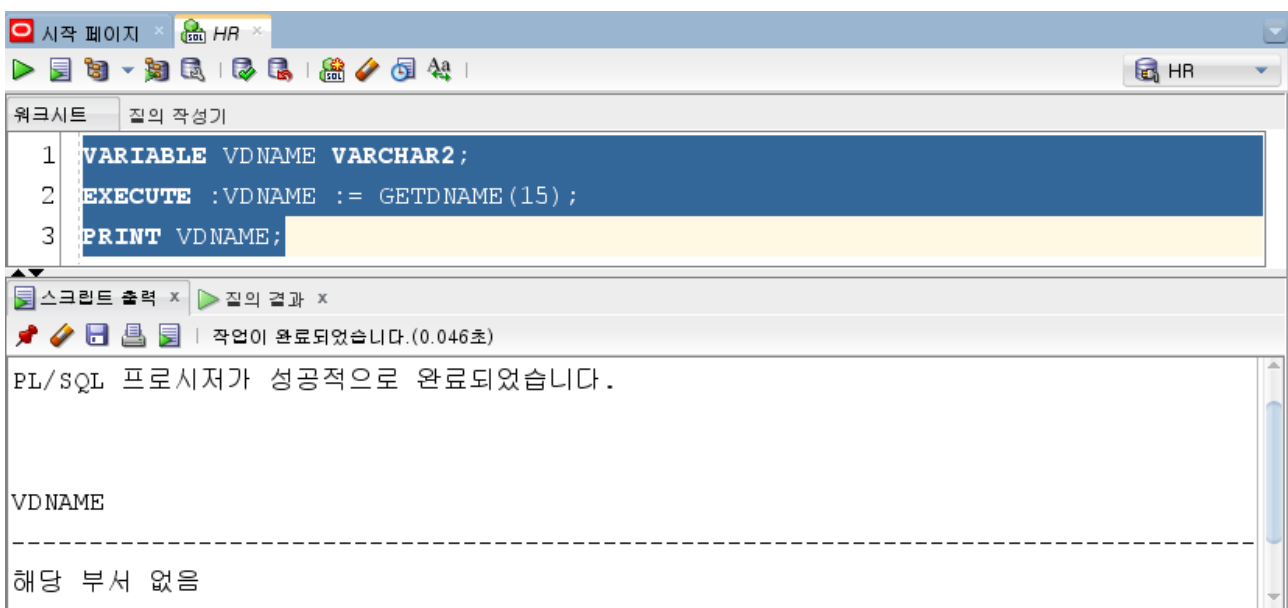
```
SELECT EMPLOYEE_ID, FIRST_NAME, TO_CHAR(HIRE_DATE, 'YYYY-MM-DD') HIREDATE,
       GETDNAME(DEPARTMENT_ID) DEPT_NAME
FROM EMPLOYEES;
```

EMPLOYEE_ID	FIRST_NAME	HIREDATE	DEPT_NAME
1	198 Donald	2007-06-21	Shipping
2	199 Douglas	2008-01-13	Shipping
3	200 Jennifer	2003-09-17	Administration
4	201 Michael	2004-02-17	Marketing
5	202 Pat	2005-08-17	Marketing
6	203 Susan	2002-06-07	Human Resources
7	204 Hermann	2002-06-07	Public Relations
8	205 Shelley	2002-06-07	Accounting
9	206 William	2002-06-07	Accounting
10	100 Steven	2003-06-17	Executive
11	101 Neena	2005-09-21	Executive
12	102 Lex	2001-01-13	Executive
13	103 Alexander	2006-01-03	IT
14	104 Bruce	2007-05-21	IT
15	105 David	2005-06-25	IT

```
SELECT GETDNAME(178) FROM DUAL;
```

GETDNAME(178)
1 해당 부서 없음

```
VARIABLE VNAME VARCHAR2;
EXECUTE :VNAME := GETDNAME(15);
PRINT VNAME;
```



The screenshot shows the Oracle SQL Developer interface. The top pane displays the following SQL script:

```
1 VARIABLE VNAME VARCHAR2;
2 EXECUTE :VNAME := GETDNAME(15);
3 PRINT VNAME;
```

The bottom pane shows the execution results:

```
PL/SQL 프로시저가 성공적으로 완료되었습니다.

VNAME
-----
해당 부서 없음
```

-- 부서번호를 매개변수로 해당 부서의 평균 급여를 반환하는 함수를 생성하자.

```
CREATE OR REPLACE FUNCTION GETAVGDEPT(V_DEPTNO IN EMPLOYEES.DEPARTMENT_ID%TYPE)
RETURN VARCHAR2
IS
    V_AVGSAL VARCHAR2(50);
BEGIN
    SELECT TO_CHAR(ROUND(AVG(SALARY)), '99999999') INTO V_AVGSAL
```

```

FROM EMPLOYEES
WHERE DEPARTMENT_ID = V_DEPTNO;
RETURN V_AVGSAL;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        V_AVGSAL := '해당 부서 없음';
        RETURN V_AVGSAL;
END;
/

```

-- 직원번호를 조건으로 직원의 이름, 급여, 부서명 및 부서 평균 급여를 출력해 주세요.

```

SELECT      FIRST_NAME,      SALARY,      GETDNAME(DEPARTMENT_ID)      DEPARTMENT_NAME,
GETAVGDEPT(DEPARTMENT_ID) AVG_SALARY
FROM EMPLOYEES
WHERE EMPLOYEE_ID = 190;

```

	FIRST_NAME	SALARY	DEPARTMENT_NAME	AVG_SALARY
1	Timothy	2900	Shipping	3476

[예제]

1. 오라클에서 성적처리 테이블을 생성하라.

desc sung

NO	칼럼명	자료형	크기	유일키	NULL허용	키	비고
1	hakbun	number	4	Y	N	PK	학번
2	hakname	VARCHAR2	20	N	N		이름
3	kor	number	4	N	N		국어
4	eng	number	4	N	N		영어
5	mat	number	4	N	N		수학
6	tot	number	4	N	Y		총합
7	ave	number	5,1	N	Y		평균
8	rank	number	4	N	Y		등수

2. 테이블에 학번, 이름, 국어, 영어, 수학 점수를 입력하면 총점과 평균이 자동 계산되어 입력되도록 프로시저(SUNG_INPUT)를 작성하라.

	HAKBUN	HAKNAME	KOR	ENG	MAT	TOT	AVE	RANK
1	1	홍길동	99	80	85	264	88	(null)

3. 테이블에 학번, 이름, 국어, 영어, 수학 점수를 입력하면 총점과 평균이 자동 계산되도록 트리거 (SUNGAL_TRG)를 작성하라.

	HAKBUN	HAKNAME	KOR	ENG	MAT	TOT	AVE	RANK
1	1	홍길동	99	80	85	264	88	(null)
2	2	김희진	95	84	79	258	86	(null)
3	3	이현수	83	89	99	271	90.3	(null)

4. 등수(SUNG_RANK)를 구하는 저장프로시저를 작성하고 이를 호출하여 등수가 제대로 구해지는지 확인 하자. 다음은 등수를 구하는 저장프로시저 SP_RANK가 성공적으로 작성되었다는 가정 하에 실습한 결과 이다.

```
EXECUTE SUNG_RANK
SELECT * FROM SUNG ORDER BY RANK ASC, KOR DESC, ENG DESC, MAT DESC;
```

	HAKBUN	HAKNAME	KOR	ENG	MAT	TOT	AVE	RANK
1	4	김철수	99	83	89	271	90.3	1
2	3	이현수	83	89	99	271	90.3	1
3	1	홍길동	99	80	85	264	88	3
4	2	김희진	95	84	79	258	86	4
5	5	조현정	80	75	88	243	81	5