

14. PL/SQL

자주 사용하는 명령을 PL/SQL 구문을 이용하여 데이터베이스에 저장하고 필요할 때 호출하여 사용하는 방법을 학습한다.

질의 수행 결과 반환되는 여러 개의 행을 처리하기 위해서 커서를 학습한다.

(1) PL/SQL

PL/SQL 은 Oracle's Procedural Language extension to SQL의 약자이다

SQL문장에서 변수정의, 조건처리(IF), 반복처리(LOOP, WHILE, FOR) 등을 지원한다.

다른 프로그래밍 언어와 다른 점은 **PL/SQL**은 DB에 직접 탑재되어 컴파일되고 실행되어 성능 면에서 우수하고, DB 관련 처리를 할 때 수많은 기능을 제공한다.

- PL/SQL은 SQL에 없는 기능이 제공한다.
 - 변수 선언을 할 수 있다.
 - 비교 처리를 할 수 있다.
 - 반복 처리를 할 수 있다.

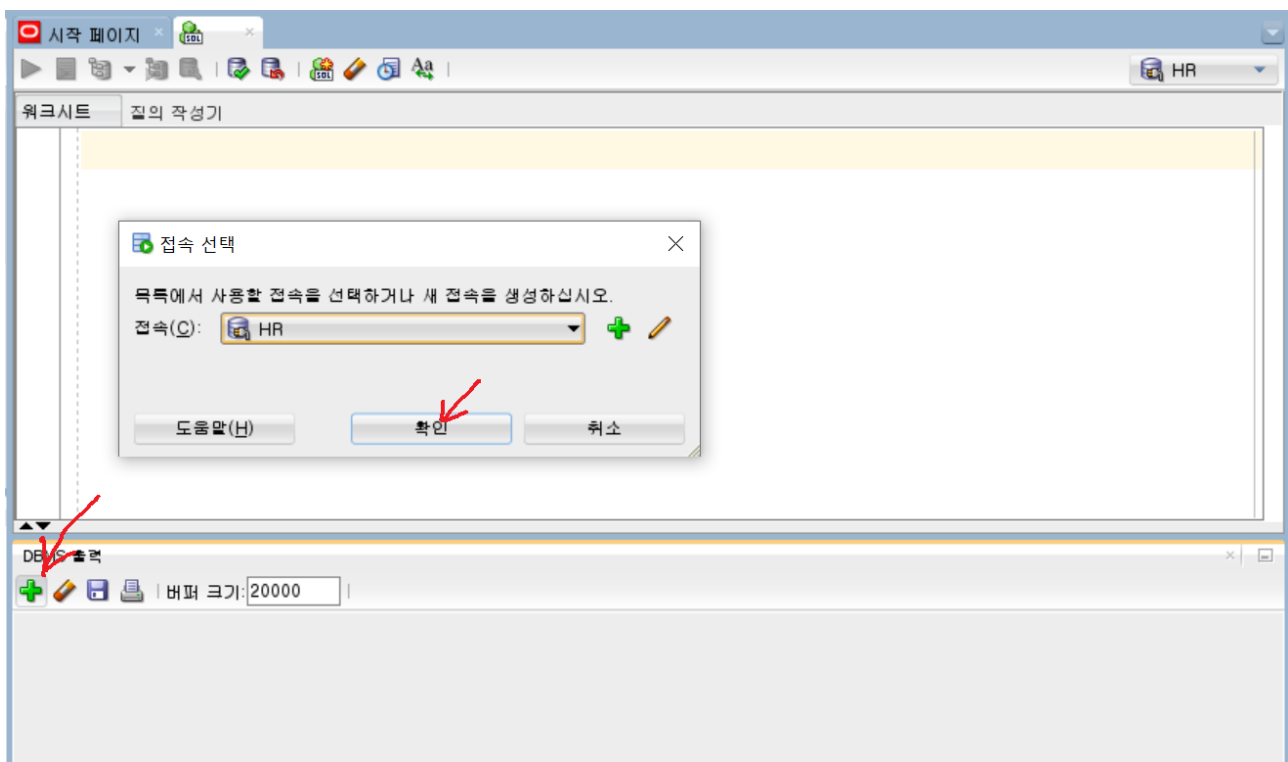
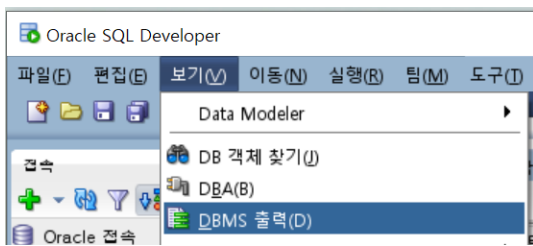
PL/SQL 소스 프로그램의 기본 단위를 블록(BLOCK)이라고 하는데, 블록은 선언부, 실행부, 예외처리부로 구성된다. 이 블록은 다시 이름이 없는 블록과 이름이 있는 블록으로 구분할 수 있는데 전자에 속하는 것이 익명블록이며, 함수, 프로시저, 패키지 등이 후자에 속한다.

장점	
<p>프로그램의 모듈화 특정 프로그램을 Procedure, function, package등의 프로그램으로 만들어 여러 응용 프로그램을 만들 시에 공통으로 이용 및 관리할 수 있다.</p> <p>관리의 용이 네트워크 입출력 감소 PL/SQL문을 사용하게 되면 모든 SQL문 및 논리적 프로그램을 프로시저 등으로 작성하며 클라이언트는 네트워크 연결을 통한 SQL문 및 프로그램 언어를 보내는 것이 아니라 프로시저 등을 실행하는 구문만 데이터베이스에 보낼 수 있어 네트워크 입출력을 현격하게 줄일 수 있다.</p>	<pre> DECLARE 선언부(DECLARE SECTION) - 변수나 상수를 선언 BEGIN 실행부(EXECUTABLE SECTION) SQL 문 제어문, 반복문 커서 EXCEPTION 예외 처리부(EXCEPTION SECTION) END; / </pre>

- PL/SQL 블록 내에서는 한 문장이 종료할 때마다 세미콜론(;)을 사용한다.
- END 뒤에 ;을 사용하여 하나의 블록이 끝났다는 것을 명시한다.
- DECLARE나 BEGIN이라는 키워드로 PL/SQL 블록이 시작하는 것을 알 수 있다.
- 단일 행 주석은 --이고 여러 행 주석 /* */이다.
- PL/SQL 블록은 행에 /가 있으면 종료된다.

모든 문장의 종결 기호는 세미콜론으로 명시해야 한다. **대입 연산자로는 :=을 사용한다.** 대입 연산자는 변수의 선언 시 및 변수의 대입에 이용된다

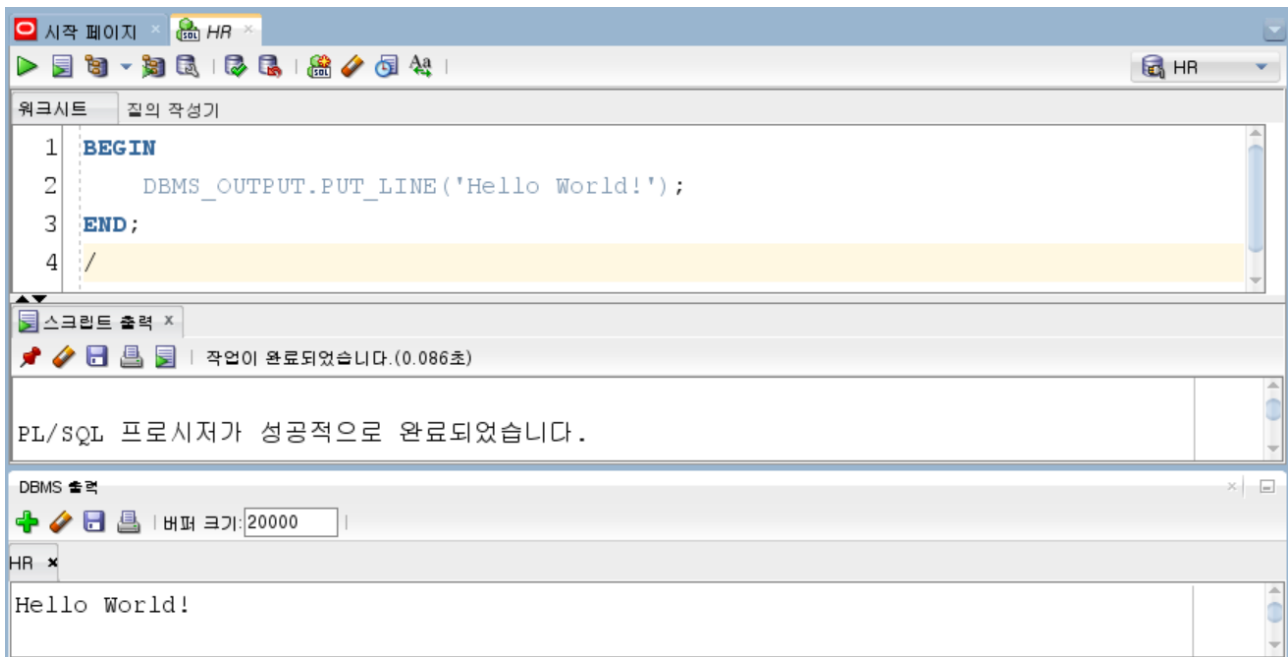
SELECT문에 의해 추출되는 DATA는 INTO절의 변수에 저장해서 처리한다.



'Hello World!'를 출력하는 PL/SQL 문을 작성한다.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
/
```

화면에 출력하기 위해서 PUT_LINE 프로시저를 이용한다. PUT_LINE은 오라클이 제공하는 프로시저로 DBMS_OUTPUT 패키지에 묶여 있다.



1) 변수

identifier [CONSTANT] datatype [NOT NULL] [:= | DEFAULT expression];

구 문	설 명
identifier	변수의 이름
CONSTANT	변수의 값을 변경할 수 없도록 제약한다.
datatype	자료형(데이터 타입)을 기술한다. - 변수 데이터 타입 : SQL 타입과 PL/SQL 타입 - PL/SQL 데이터 타입 : BOOLEAN, BINARY_INTEGER
NOT NULL	값을 반드시 포함하도록 하기 위해 변수를 제약한다.
Expression	Literal, 다른 변수, 연산자나 함수를 포함하는 표현식

VEMPNO NUMBER(4);

PL/SQL에서 변수를 선언할 때 사용되는 자료형은 SQL에서 사용하던 자료형과 거의 유사하다. 숫자를 저장하려면 NUMBER를 사용하고 문자를 저장하려면 VARCHAR2를 사용해서 선언한다. 상수는 다음과 같이 선언하면 된다. 변수와는 달리 한 번 값을 할당하면 변하지 않는다

상수명 CONSTANT 데이터 타입 := 상수값;

① 변수의 값 대입

변수의 값을 지정하거나 재지정하기 위해 PL/SQL의 지정문자를 사용한다.

대입(지정) 연산자(:=)는 좌측에 존재하는 변수에 우측 값을 대입한다는 의미이다.

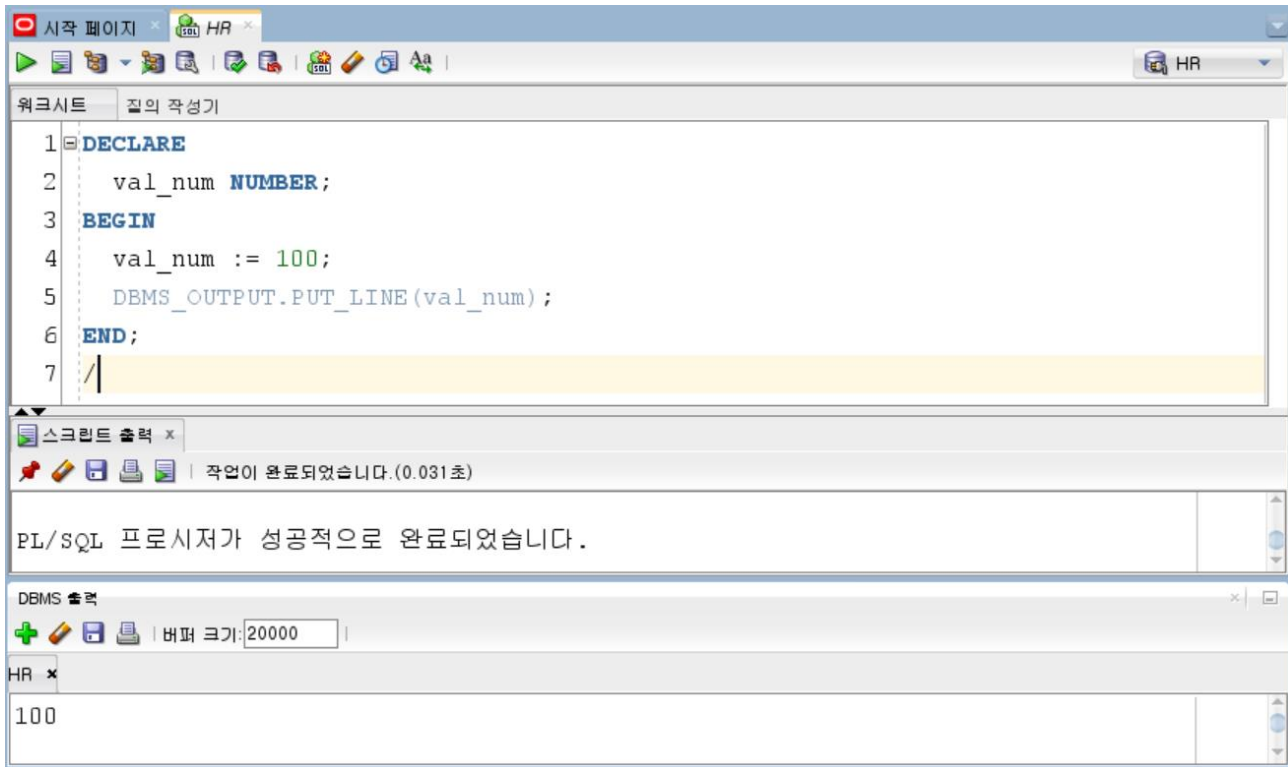
identifier := expression;

DECLARE

```

    val_num NUMBER;
BEGIN
    val_num := 100;
    DBMS_OUTPUT.PUT_LINE(val_num);
END;
/

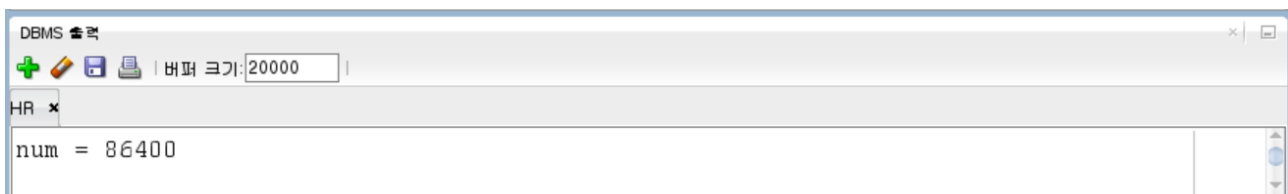
```



```

DECLARE
    num NUMBER := 24*60*60;
BEGIN
    DBMS_OUTPUT.PUT_LINE('num = ' || TO_CHAR(num));
END;
/

```



변수의 선언 및 할당을 하고 그 변수 값을 출력한다.

```

DECLARE
    VEMPNO NUMBER(4);
    VENAME VARCHAR2(10);

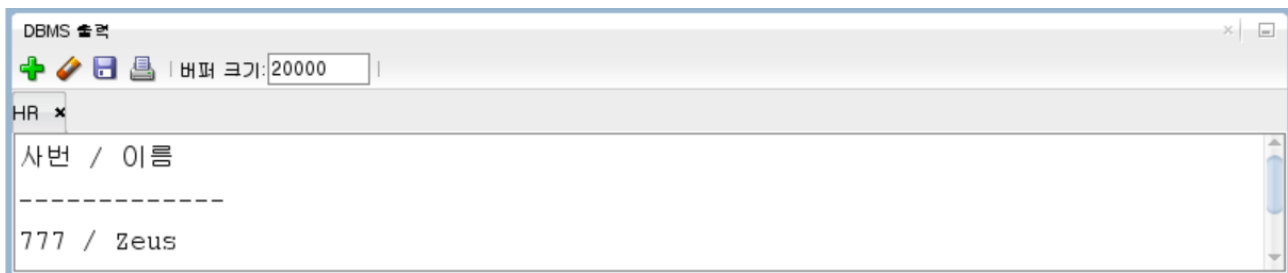
```

```

BEGIN
    VEMPNO := 777;
    VENAME := 'Zeus';

    DBMS_OUTPUT.PUT_LINE('사번 / 이름');
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE(VEMPNO || ' / ' || VENAME);
END;
/

```



② 스칼라 변수/레퍼런스 변수

PL/SQL에서 변수를 선언하기 위해 사용할 수 있는 데이터형은 크게 스칼라와 레퍼런스로 나눈다.

• 스칼라

SQL에서의 자료형 지정과 거의 동일하다.

```

VEMPNO NUMBER(4);
VENAME VARCHAR2(10);

```

• 레퍼런스 : %TYPE 속성과 %ROWTYPE 속성 사용한다.

이전에 선언된 다른 변수 또는 데이터베이스의 칼럼에 맞추어 변수를 선언하기 위해 %TYPE 속성을 사용한다.

```

VEMPNO EMPLOYEES.EMPLOYEE_ID%TYPE;
VENAME EMPLOYEES.FIRST_NAME%TYPE;

```

%TYPE 속성을 사용하여 선언한 VEMPNO 변수는 EMP테이블의 EMPNO칼럼의 자료형과 크기를 그대로 참조해서 정의한다.

%ROWTYPE은 로우 단위로 참조한다.

```

VEMP EMPLOYEES%ROWTYPE;

```

%ROWTYPE을 사용 시 장점은 특정 테이블의 칼럼의 개수와 데이터 형식을 모르더라도 지정할 수 있다. SELECT 문장으로 로우를 검색할 때 유리하다.

③ PL/SQL에서 SQL문장

PL/SQL의 SELECT 문은 INTO절이 필요한데, INTO절에는 데이터를 저장할 변수를 기술한다.

SELECT 절에 있는 칼럼은 INTO절에 있는 변수와 1대1대응을 하기에 개수와 데이터의 형, 길이가 일

치하여야 한다.

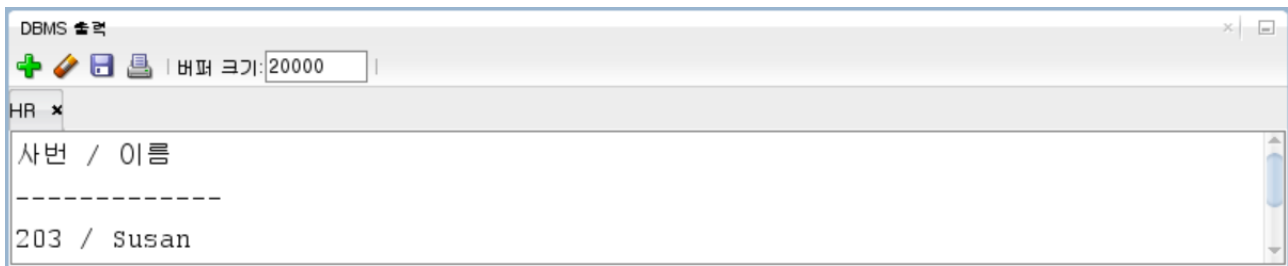
```
SELECT select_list
INTO {variable_name1 [variable_name2,...] | record_name}
FROM table_name
WHERE condition;
```

구문	설 명
select_list	열의 목록이며 행 함수, 그룹 함수, 표현식을 기술할 수 있다.
variable_name	읽어들인 값을 저장하기 위한 스칼라 변수
record_name	읽어 들인 값을 저장하기 위한 PL/SQL RECORD 변수
Condition	PL/SQL 변수와 상수를 포함하여 열명, 표현식, 상수, 비교 연산자로 구성되며 오직 하나의 값을 RETURN할 수 있는 조건이어야 한다.

```
SELECT EMPLOYEE_ID, FIRST_NAME INTO VEMPNO, VENAME
FROM EMPLOYEES
WHERE FIRST_NAME='Susan';
```

VEMPNO, VENAME 변수는 칼럼(EMPLOYEE_ID, FIRST_NAME)과 동일한 데이터형을 갖도록 하기 위해서 %TYPE 속성을 사용한다. INTO 절의 변수는 SELECT에서 기술한 칼럼의 데이터형뿐만 아니라 칼럼의 수와도 일치해야 한다.

```
DECLARE
    -- %TYPE 속성으로 칼럼 단위로 데이터를 저장할 수 있는 레퍼런스 변수 선언
    VEMPNO EMPLOYEES.EMPLOYEE_ID%TYPE;
    VENAME EMPLOYEES.FIRST_NAME%TYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('사번 / 이름');
    DBMS_OUTPUT.PUT_LINE('-----');
    -- SELECT 문을 수행한 결과 값이 INTO 뒤에 기술한 변수에 저장된다.
    SELECT EMPLOYEE_ID, FIRST_NAME INTO VEMPNO, VENAME
    FROM EMPLOYEES
    WHERE FIRST_NAME='Susan';
    -- 레퍼런스 변수에 저장된 값을 출력한다.
    DBMS_OUTPUT.PUT_LINE(VEMPNO || ' / ' || VENAME);
END;
/
```



④ PL/SQL TABLE TYPE

PL/SQL TABLE TYPE은 로우에 대해 배열처럼 액세스하기 위해 사용한다. 배열과 유사하고 PL/SQL 테이블을 액세스하기 위해 BINARY_INTEGER 데이터형을 기준으로 PL/SQL 테이블 요소를 저장하는 스칼라 또는 레코드 데이터형의 칼럼을 포함해야 한다. 또한 이들은 동적으로 자유롭게 증가할 수 있다.

```
TYPE table_type_name IS TABLE OF
    { column_type | variable%TYPE | table.column%TYPE } {NOT NULL}
[INDEX BY BINARY_INTEGER];
identifier table_type_name;
```

구분	설 명
table_type_name	테이블형의 이름
column_type	VARCHAR2, DATE, NUMBER과 같은 스칼라 데이터형
identifier	전체 PL/SQL 테이블을 나타내는 식별자 이름

TABLE 변수를 사용하여 EMP 테이블에서 이름과 업무를 출력한다.

```
DECLARE
    -- 테이블 타입을 정의
    TYPE first_name_table_type IS TABLE OF EMPLOYEES.FIRST_NAME%TYPE
    INDEX BY BINARY_INTEGER;
    TYPE job_id_table_type IS TABLE OF EMPLOYEES.JOB_ID%TYPE
    INDEX BY BINARY_INTEGER;
    TYPE employee_id_table_type IS TABLE OF EMPLOYEES.EMPLOYEE_ID%TYPE
    INDEX BY BINARY_INTEGER;

    -- 테이블 타입으로 변수 선언
    name_table first_name_table_type;
    job_table job_id_table_type;
    empno_table employee_id_table_type;
    i BINARY_INTEGER := 0;
BEGIN
    -- EMPLOYEES 테이블로부터 사원 이름과 직급을 얻어온다.
    FOR k IN (SELECT EMPLOYEE_ID, FIRST_NAME, JOB_ID FROM EMPLOYEES) LOOP
        i := i + 1;          -- 인덱스를 증가시켜가며
```

```

        name_table(i) := k.FIRST_NAME;      -- 테이블에 얻어온 사원 이름과
        job_table(i) := k.JOB_ID;           -- 직급을 배열처럼 저장한다.
        empno_table(i):=k.EMPLOYEE_ID;

    END LOOP;

    -- 테이블에 저장된 내용을 출력
    DBMS_OUTPUT.PUT_LINE('사번' || ' / ' || RPAD('이름', 10) || ' / ' || RPAD('직무', 9));
    DBMS_OUTPUT.PUT_LINE('-----');
    FOR j IN 1..i LOOP
        DBMS_OUTPUT.PUT_LINE(empno_table(j) || ' / ' || RPAD(name_table(j), 10) || ' / '
                               || RPAD(job_table(j), 9));
    END LOOP;

END;
/

```

```

DBMS 출력
버퍼 크기: 20000
HR x
사번 / 이름      / 직무
-----
198 /Donald      /SH_CLERK
199 /Douglas     /SH_CLERK
200 /Jennifer    /AD_ASST
201 /Michael     /MK_MAN
202 /Pat         /MK_REP
203 /Susan       /HR_REP
204 /Hermann     /PR_REP
205 /Shelley     /AC MGR

```

⑤ PL/SQL RECORD TYPE

PL/SQL RECORD TYPE은 프로그램 언어의 구조체와 유사하다. PL/SQL **RECORD**는 **FIELD(ITEM)**들의 집합을 하나의 논리적 단위로 처리할 수 있게 해주므로 테이블의 ROW를 읽어올 때 편리하다.

TYPE type_name IS RECORD

(field_name1 {scalar_datatype|record_type} [NOT NULL] [(:= | DEFAULT) expr],
 field_name2 {scalar_datatype|record_type} [NOT NULL] [(:= | DEFAULT) expr],. . .);

identifier type_name;

구문	설 명
type_name	RECODE 형의 이름, 이 식별자는 RECODE를 선언하기 위해 사용한다.
field_name	RECODE내의 필드명

개별 필드를 참조하거나 초기화하기 위해 "."을 사이에 두고 레코드 이름과 필드 이름을 기술한다.

Record_name.field_name

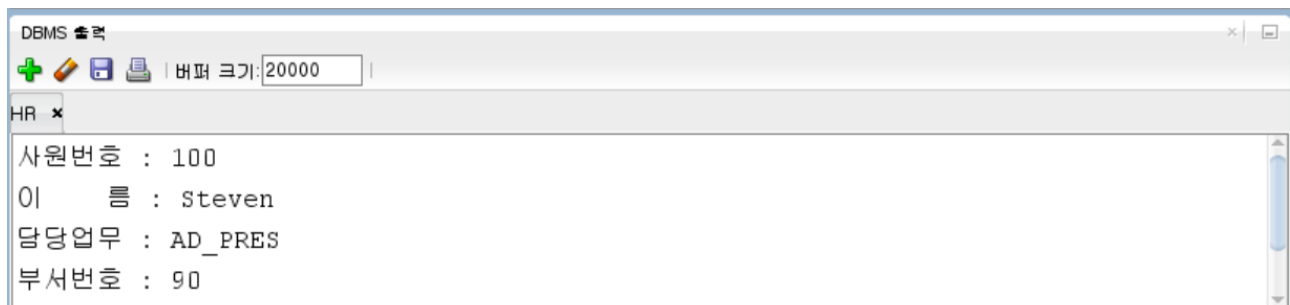
EMPLOYEES 테이블에서 SCOTT 사원의 정보를 출력하라

```
DECLARE
    -- 레코드 타입을 정의
    TYPE emp_record_type IS RECORD(
        v_empno EMPLOYEES.EMPLOYEE_ID%TYPE,
        v_ename EMPLOYEES.FIRST_NAME%TYPE,
        v_job EMPLOYEES.JOB_ID%TYPE,
        v_deptno EMPLOYEES.DEPARTMENT_ID%TYPE
    );

    -- 레코드로 변수 선언
    emp_record emp_record_type;
BEGIN
    -- SCOTT 사원의 정보를 레코드 변수에 저장
    SELECT EMPLOYEE_ID, FIRST_NAME, JOB_ID, DEPARTMENT_ID
    INTO emp_record
    FROM EMPLOYEES
    WHERE MANAGER_ID IS NULL;

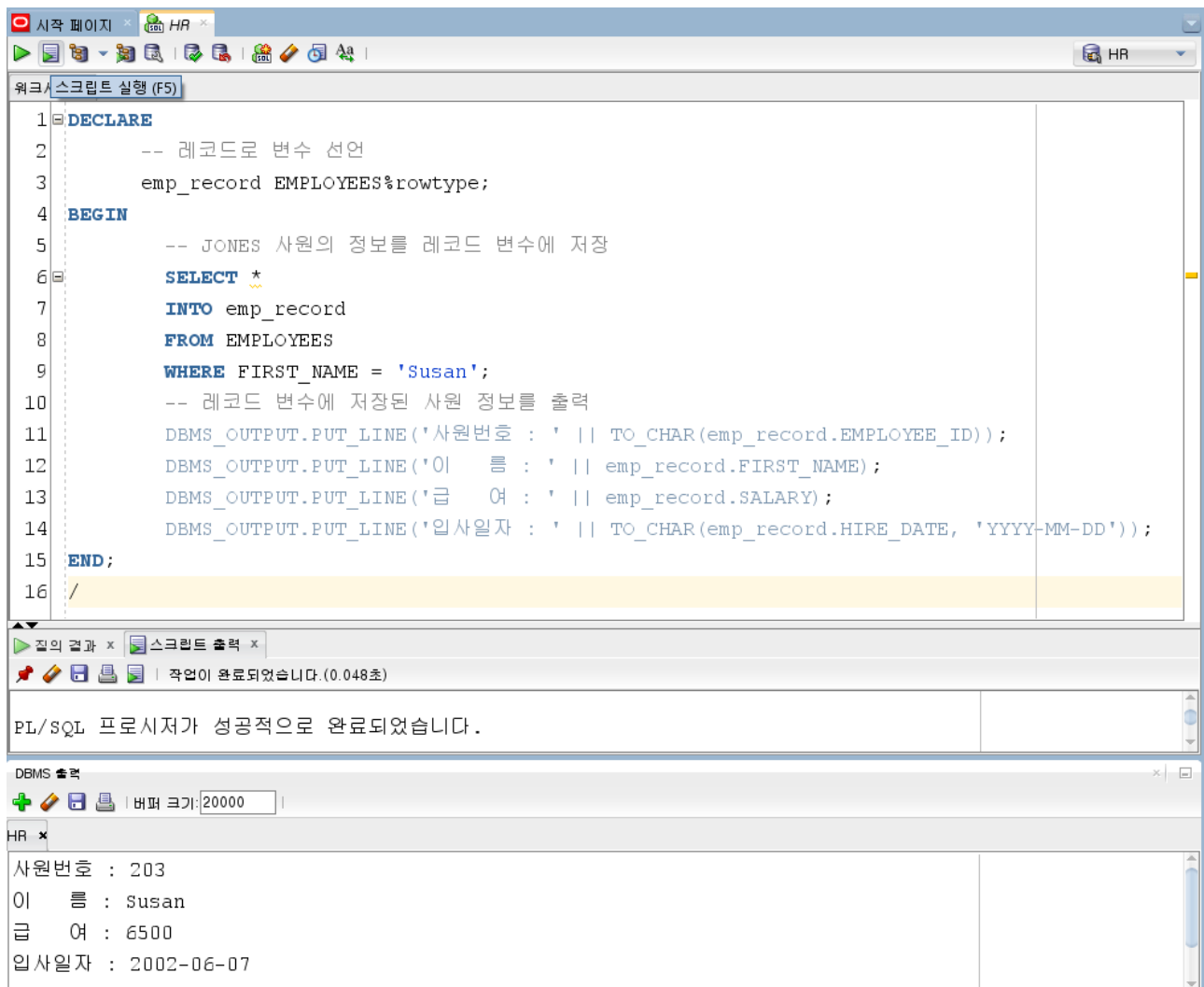
    -- 레코드 변수에 저장된 사원 정보를 출력
    DBMS_OUTPUT.PUT_LINE('사원번호 : ' || TO_CHAR(emp_record.v_empno));
    DBMS_OUTPUT.PUT_LINE('이      름 : ' || emp_record.v_ename);
    DBMS_OUTPUT.PUT_LINE('담당업무 : ' || emp_record.v_job);
    DBMS_OUTPUT.PUT_LINE('부서번호 : ' || TO_CHAR(emp_record.v_deptno));

END;
/
```



전체 레코드를 참조하기 위해서는 %rowtype으로 선언하면 된다.

```
DECLARE
    -- 레코드로 변수 선언
    emp_record EMPLOYEES%rowtype;
BEGIN
    -- JONES 사원의 정보를 레코드 변수에 저장
    SELECT *
    INTO emp_record
    FROM EMPLOYEES
    WHERE FIRST_NAME = 'Susan';
    -- 레코드 변수에 저장된 사원 정보를 출력
    DBMS_OUTPUT.PUT_LINE('사원번호 : ' || TO_CHAR(emp_record.EMPLOYEE_ID));
    DBMS_OUTPUT.PUT_LINE('이름 : ' || emp_record.FIRST_NAME);
    DBMS_OUTPUT.PUT_LINE('급여 : ' || emp_record.SALARY);
    DBMS_OUTPUT.PUT_LINE('입사일자 : ' || TO_CHAR(emp_record.HIRE_DATE, 'YYYY-MM-DD'));
END;
/
```



The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL script with line numbers 1 through 16. The script declares a record variable, selects data for Susan, and outputs it. The output window at the bottom shows the results of the script execution.

PL/SQL 프로시저가 성공적으로 완료되었습니다.

DBMS 출력

버퍼 크기: 20000

HR

사원번호 : 203
이름 : Susan
급여 : 6500
입사일자 : 2002-06-07

[문제 1] EMPLOYEES 테이블에 등록된 총사원의 수와 급여의 합, 급여의 평균을 변수에 대입하여 출력하여 보자.

DBMS 출력

+ | 버퍼 크기: 20000

HR x

총사원의 수 : 107
급여의 합 : 691416
급여의 평균 : 6461.83

[문제 2] Clara 사원의 직무, 급여, 입사일자, 커미션, 부서명을 변수에 대입하여 출력하여 보자.

DBMS 출력

+ | 버퍼 크기: 20000

HR x

직무 : SA_REP
급여 : 10500
입사일자 : 05/11/11
커미션 : .25
부서명 : Sales

3) 제어문

PL/SQL은 여러 가지 제어 구조를 이용하여 문장들의 논리적 흐름을 변경할 수 있다.

조건에 의해 분기하는 선택문과 반복된 문장을 한번 기술하고도 여러 번 수행하도록 하는 반복문이 있다.

① IF ~ THEN ~ END IF

특정 조건을 만족하면 어떤 처리를 하고, 그렇지 않으면 아무 처리도 하지 않는다.

```
IF 조건문 THEN
    조건처리;
END IF;
```

연봉을 구하는 예제를 작성한다.

연봉 계산을 위해 급여*12+커미션이란 공식을 사용한다. COMM 칼럼 값이 NULL이면 연봉 역시 NULL 이 구해진다. 왜냐하면 NULL 값과의 연산 결과는 항상 NULL이기 때문이다. 따라서 커미션 칼럼이 NULL 값일 때는 0으로 바꿔줘야 올바른 연봉 계산을 할 수 있다.

```
DECLARE
    -- %ROWTYPE 속성으로 로우 를 저장할 수 있는 레퍼런스 변수 선언
    VEMP EMPLOYEES%ROWTYPE;
    ANNSAL NUMBER(12,2);
BEGIN
    DBMS_OUTPUT.PUT_LINE('CEO의 정보');
    DBMS_OUTPUT.PUT_LINE('-----');
    -- KING CEO의 전체 정보를 로우 단위로 얻어와 VEMP에 저장한다.
    SELECT * INTO VEMP
    FROM EMPLOYEES
    WHERE LAST_NAME='King' AND MANAGER_ID IS NULL;
    -- 커미션이 NULL 일 경우 이를 0으로 변경해야 올바른 연봉 계산이 가능하다.
    IF (VEMP.COMMISSION_PCT IS NULL) THEN
        VEMP.COMMISSION_PCT := 0;
    END IF;

    -- 스칼라 변수에 연봉을 계산할 결과를 저장한다.
    ANNSAL:=VEMP.SALARY*12+(VEMP.SALARY*VEMP.COMMISSION_PCT);
    -- 레퍼런스 변수와 스칼라 변수에 저장된 값을 출력한다.
    DBMS_OUTPUT.PUT_LINE( '직원번호: ' || VEMP.EMPLOYEE_ID || ' / 직원명 : ' ||
        VEMP.FIRST_NAME || ' / 연봉 : ' || to_char(ANNSAL,'$999,999') );
END;
/
```

DBMS 실행

버퍼 크기: 20000

HR x

CEO의 정보

직원번호: 100 / 직원명 : Steven / 연봉 : \$288,000

② IF ~ THEN ~ ELSE ~ END IF

조건을 만족할 때의 처리와 그렇지 않을 때의 처리, 즉 두 가지 처리문 중에서 한 개를 선택해야 할 경우 사용한다.

IF 조건문 THEN

조건처리1;

ELSE

조건처리2;

END IF;

DECLARE

-- %ROWTYPE 속성으로 로우를 저장할 수 있는 레퍼런스 변수 선언

VEMP EMPLOYEES%ROWTYPE;

ANNSAL NUMBER(10,2);

BEGIN

DBMS_OUTPUT.PUT_LINE('직원번호/ 이름 / 연봉');

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

-- Susan 직원의 전체 정보를 로우 단위로 얻어와 VEMP에 저장한다.

SELECT * INTO VEMP FROM EMPLOYEES

WHERE FIRST_NAME='Susan';

IF (VEMP.COMMISSION_PCT IS NULL) THEN -- 커미션이 NULL이면

ANNSAL:=VEMP.SALARY*12;

ELSE -- 커미션이 NULL이 아니면

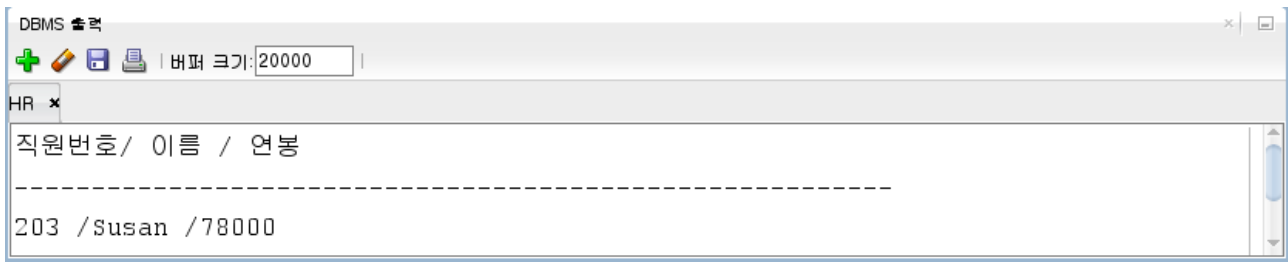
ANNSAL:=VEMP.SALARY*12+(VEMP.SALARY*VEMP.COMMISSION_PCT);

END IF;

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

END;

/



③ IF ~ THEN ~ ELSIF ~ ELSE ~ END IF

여러 개 조건에 따라 처리도 여러 개일 때 사용하는 다중 IF 문이다.

IF 조건문 THEN

조건처리1;

ELSIF condition THEN

조건처리2;

ELSIF condition THEN

조건처리3;

ELSE

조건처리n;

END IF

DECLARE

-- %ROWTYPE 속성으로 로우 를 저장할 수 있는 레퍼런스 변수 선언

VEMP EMPLOYEES%ROWTYPE;

VDNAME DEPARTMENTS.DEPARTMENT_NAME%TYPE;

BEGIN

DBMS_OUTPUT.PUT_LINE('직원번호 / 이름 / 부서번호 / 부서명');

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

-- 직원번호가 203 사원의 전체 정보를 로우 단위로 얻어와 VEMP에 저장한다.

SELECT * INTO VEMP FROM EMPLOYEES WHERE EMPLOYEE_ID=203;

IF (VEMP.DEPARTMENT_ID = 20) THEN

VDNAME := 'Marketing';

ELSIF (VEMP.DEPARTMENT_ID = 30) THEN

VDNAME := 'Purchasing';

ELSIF (VEMP.DEPARTMENT_ID = 40) THEN

VDNAME := 'Human Resources';

ELSIF (VEMP.DEPARTMENT_ID = 50) THEN

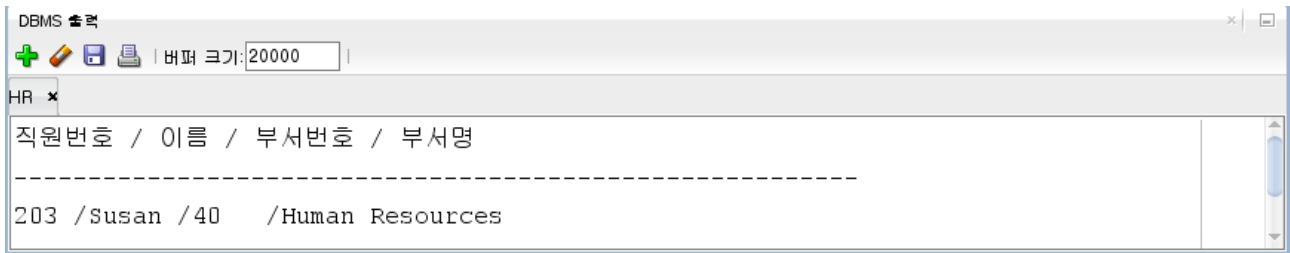
VDNAME := 'Shipping';

END IF;

DBMS_OUTPUT.PUT_LINE(VEMP.EMPLOYEE_ID || ' / ' || VEMP.FIRST_NAME || ' / ' ||

RPAD(VEMP.DEPARTMENT_ID, 4) || ' / ' || VDNAME);

```
END;
/
```

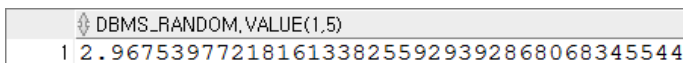


DBMS_RANDOM.VALUE

- 랜덤한 숫자를 생성한다.
- 형식 : DBMS_RANDOM.VALUE(최소 범위의 숫자, 최대 범위의 숫자)

(예제)

```
SELECT DBMS_RANDOM.VALUE(1,5) FROM DUAL;
```



DBMS_RANDOM.STRING

- 랜덤한 문자열을 생성한다.
- 형식 : DBMS_RANDOM.STRING(옵션문자, 길이수)

- 옵션문자는 아래와 같다.

'u', 'U'	대문자	'l', 'L'	소문자	'a', 'A'	대소문자 구분없는 영문자
'x', 'X'	영문자와 숫자 혼합	'p', 'P'	문자 혼합		

(예제)

```
SELECT DBMS_RANDOM.STRING('U', 1) FROM DUAL; --1개의 임의의 문자
```

```
SELECT DBMS_RANDOM.STRING('A', 2) FROM DUAL; --대소문자 관계없이 2개의 임의의 문자
```



```
DECLARE
```

```
    vsal NUMBER := 0;
```

```
    vdeptno NUMBER := 0;
```

```
BEGIN
```

```
    vdeptno := ROUND(DBMS_RANDOM.VALUE (10, 110), -1);
```

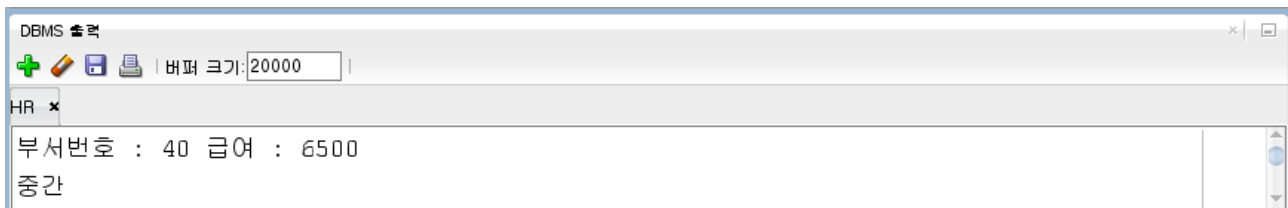
```
    SELECT SALARY INTO vsal
```

```

FROM EMPLOYEES
WHERE DEPARTMENT_ID = vdeptno AND ROWNUM = 1;
DBMS_OUTPUT.PUT_LINE('부서번호 : ' || vdeptno || ' 급여 : ' || vsal);

IF vsal BETWEEN 1 AND 5000 THEN
    DBMS_OUTPUT.PUT_LINE('낮음');
ELSIF vsal BETWEEN 5001 AND 10000 THEN
    DBMS_OUTPUT.PUT_LINE('중간');
ELSIF vsal BETWEEN 10001 AND 20000 THEN
    DBMS_OUTPUT.PUT_LINE('높음');
ELSE
    DBMS_OUTPUT.PUT_LINE('최상위');
END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(vdeptno || ' 부서에 해당 사원이 없습니다') ;
END;
/

```



④ 반복문

반복문은 SQL 문을 반복적으로 여러 번 실행하고자 할 때 사용한다.

PL/SQL에서는 다양한 반복문이 사용된다.

- 조건 없이 반복 작업을 제공하기 위한 BASIC LOOP문
- COUNT를 기본으로 작업의 반복 제어를 제공하는 FOR LOOP문
- 조건을 기본으로 작업의 반복 제어를 제공하는 WHILE LOOP문
- LOOP를 종료하기 위한 EXIT문

⑤ BASIC LOOP문

LOOP

```

    statement1;
    statement2;
    EXIT [WHEN condition];
END LOOP;

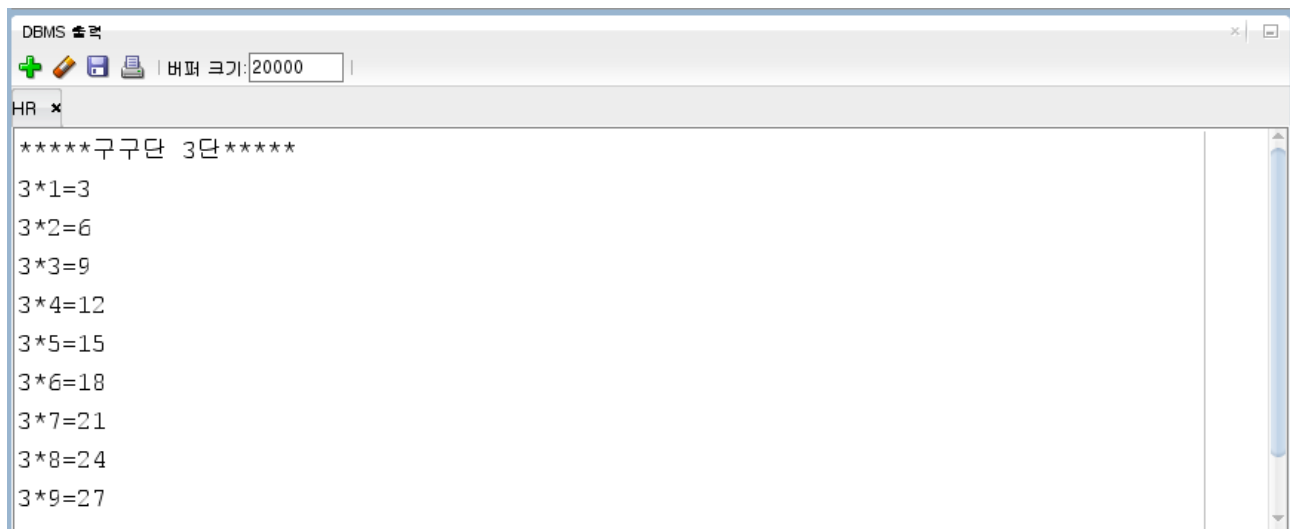
```

loop에서 end loop 사이를 계속 반복해서 실행한다. 이러한 루프를 무한 루프라 하며, 여기서 빠져 나가려면 exit문을 사용한다. exit문을 이용하여 루프를 종료할 수 있다. 조건에 따라 루프를 종료할 수 있도록 WHEN 절을 덧붙일 수 있다.

실행 상의 흐름이 END LOOP에 도달할 때마다 그와 짝을 이루는 LOOP 문으로 제어가 되돌아간다.

여기서 빠져나가려면 EXIT문을 사용한다.

```
DECLARE
    vn_base_num NUMBER := 3;
    vn_cnt       NUMBER := 1;
BEGIN
    DBMS_OUTPUT.PUT_LINE ('*****구구단 3단*****');
    LOOP
        DBMS_OUTPUT.PUT_LINE (vn_base_num || '*' || vn_cnt || '=' || vn_base_num * vn_cnt);
        vn_cnt := vn_cnt + 1;
        --EXIT WHEN vn_cnt >9;
        IF vn_cnt >9 THEN
            EXIT;
        END IF;
    END LOOP;
END;
/
```



```
DBMS 출력
버퍼 크기: 20000
*****구구단 3단*****
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
```

⑥ FOR LOOP문

FOR LOOP는 반복되는 횟수가 정해진 반복문을 처리한다.

FOR LOOP문에서 사용되는 인덱스는 정수로 자동 선언되므로 따로 선언할 필요가 없다.

FOR LOOP문은 LOOP를 반복할 때마다 자동적으로 1씩 증가 또는 감소한다. REVERSE는 1씩 감소함을 의미한다.

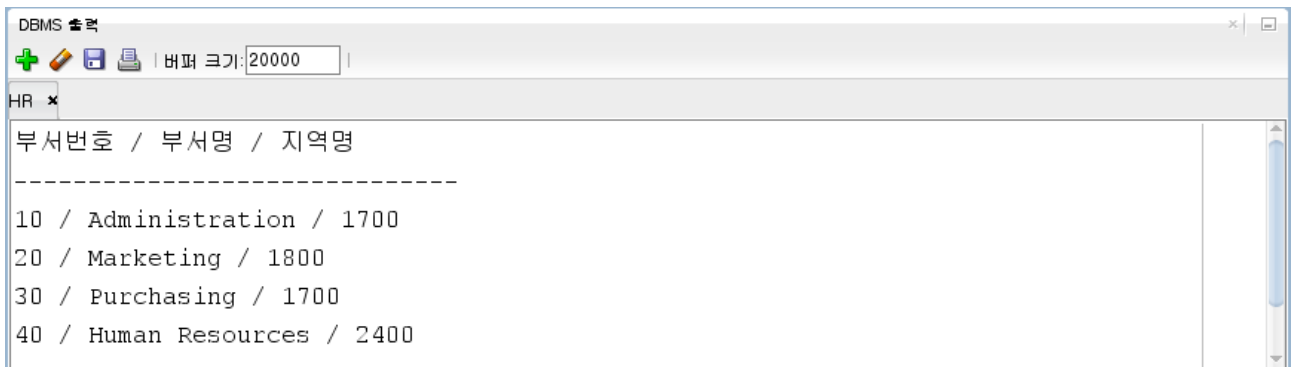
```
FOR index_counter IN [REVERSE] lower_bound..upper_bound LOOP
    statement1;
    statement2;
END LOOP;
```

구 문	설 명
index_counter	lower_bound나 upper_bound에 도달할 때까지 LOOP를 반복함으로써 1씩 자동적으로 증가하거나 감소되는 값을 가진 암시적으로 선언된 정수이다.
REVERSE	lower_bound 에서 upper_bound까지 반복함으로써 인덱스가 1씩 감소되도록 한다.
lower_bound	index_counter 값의 범위에 대한 하단 바운드 값을 지정한다.
upper_bound	index_counter 값의 범위에 대한 상단 바운드 값을 지정한다.

```

DECLARE
    VDEPT DEPARTMENTS%ROWTYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호 / 부서명 / 지역명');
    DBMS_OUTPUT.PUT_LINE('-----');
    -- 변수 CNT는 1부터 1씩 증가하다가 4에 도달하면 반복문에서 벗어난다.
    FOR CNT IN 1..4 LOOP
        SELECT * INTO VDEPT FROM DEPARTMENTS
        WHERE DEPARTMENT_ID=10*CNT;
        DBMS_OUTPUT.PUT_LINE(VDEPT.DEPARTMENT_ID || ' / ' || VDEPT.DEPARTMENT_NAME || '
/ ' || VDEPT.LOCATION_ID);
    END LOOP;
END;
/

```



⑦ WHILE LOOP문

제어 조건이 TRUE인 동안만 일련의 문장을 반복하기 위해 WHILE LOOP 문장을 사용한다. 조건은 반복이 시작될 때 체크하게 되어 LOOP내의 문장이 한 번도 수행되지 않을 경우도 있다. LOOP을 시작할 때 조건이 FALSE이면 반복 문장을 탈출하게 된다.

WHILE 조건문 LOOP

```

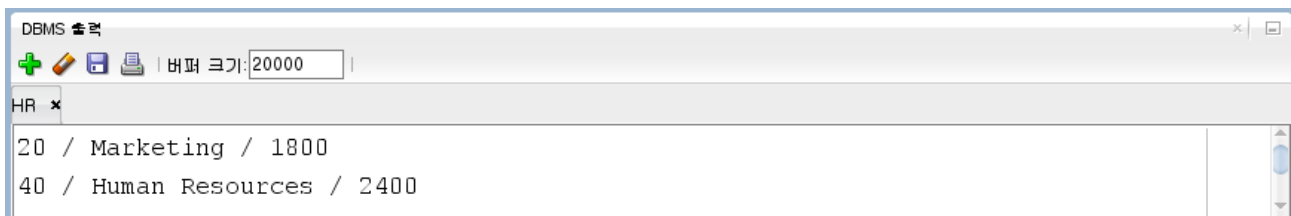
    statement1;
    statement2;
END LOOP;

```

```

DECLARE
  I NUMBER := 1;
  VDEPT DEPARTMENTS%ROWTYPE;
BEGIN
  WHILE I <= 4 LOOP
    IF I MOD 2 = 0 THEN
      SELECT * INTO VDEPT FROM DEPARTMENTS WHERE DEPARTMENT_ID =10*I;
      DBMS_OUTPUT.PUT_LINE(VDEPT.DEPARTMENT_ID || ' / ' || VDEPT.DEPARTMENT_NAME || ' / ' ||
VDEPT.LOCATION_ID);
    END IF;
    I := I+1;
  END LOOP;
END;
/

```



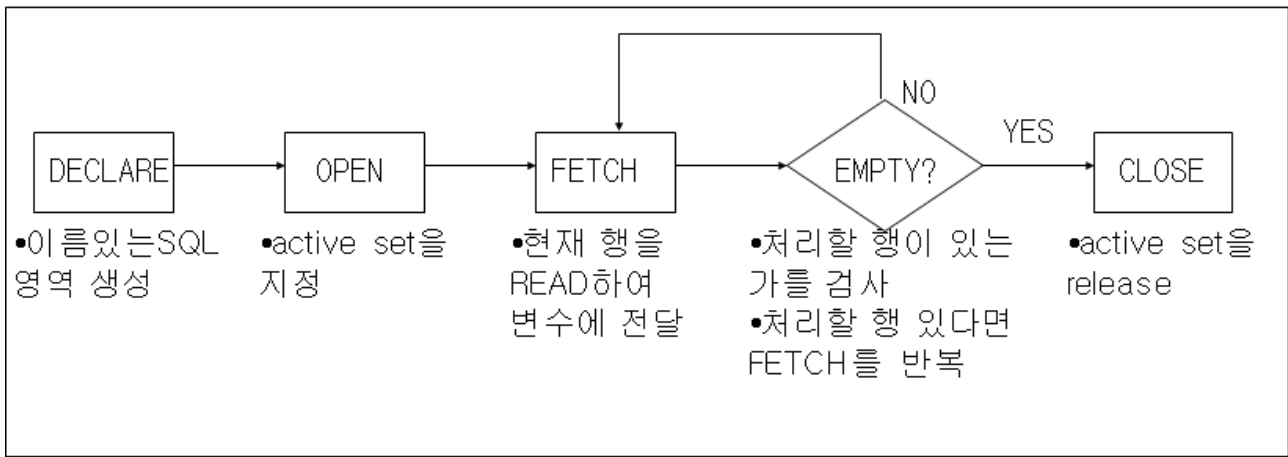
⑧ 커서

커서란 특정 SQL 문장을 처리한 결과를 담고 있는 영역을 가리키는 일종의 포인터로, 커서를 사용하면 처리된 SQL 문장의 결과 집합에 접근할 수 있다. 커서의 종류에는 묵시적 커서와 명시적 커서가 있다. 묵시적 커서란 오라클 내부에 자동으로 생성하는 커서, 명시적 커서는 사용자가 직접 정의해서 사용하는 커서를 말한다. 커서는 “커서 열기(open) – 패치(fetch) – 커서닫기(close)” 3단계로 진행된다.

```

DECLARE
  CURSOR cursor_name IS statement;           -- 커서 선언
BEGIN
  OPEN cursor_name;                             -- 커서 열기
  --커서로부터 데이터를 읽어와 변수에 저장
  FECTCH cursor_name INTO variable_name;
  CLOSE cursor_name;                           -- 커서 닫기
END;

```



• DECLARE CURSOR (커서 선언)

사용할 커서를 선언부에 직접 정의해야 한다. 사용할 커서에 이름을 부여하고 이 커서에 대한 쿼리를 선언해야 한다.

형식 **CURSOR cursor_name IS statement;**

예 **CURSOR C1 IS SELECT * FROM DEPARTMENTS;**

• OPEN CURSOR(커서 열기)

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

형식 **OPEN cursor_name;**

예 OPEN C1;

• FETCH CURSOR(패치 단계에서 커서 사용)

정의한 커서를 열고 난 후에야 SELECT문의 결과로 반환되는 로우에 접근할 수 있다.

결과 집합의 로우 수는 보통 1개 이상이므로 전체 로우에 접근하기 위해서는 반복문을 사용해야 한다.

- FETCH문은 결과 셋에서 로우 단위로 데이터를 읽어 들임
- FETCH 후에 CURSOR는 결과 셋에서 다음 행으로 이동

형식 **FETCH cursor_name
INTO {variable1 [,variable2,]};**

예 LOOP

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

EXIT WHEN C1%NOTFOUND;

END LOOP;

얻어진 여러 개의 행에 대한 결과값을 모두 처리하려면 반복문에 FETCH문을 기술해야 한다. NOTFOUND는 커서의 상태를 알려주는 속성 중에 하나인데 커서 영역의 자료가 모두 FETCH 되었다면 TRUE를 되돌린다. 즉 커서 C1 영역의 자료가 모두 FETCH되면 반복문을 탈출하게 된다.

- 커서의 상태

속 성	의 미
%NOTFOUND	커서 영역의 자료가 모두 FETCH 되었다면 TRUE

%FOUND	커서 영역에 FETCH 되지 않은 자료가 있다면 TRUE
%ISOPEN	커서가 OPEN된 상태이면 TRUE
%ROWCOUNT	커서가 얻어 온 레코드의 개수

- CLOSE CURSOR(커서 닫기)

CLOSE 문장은 CURSOR를 사용할 수 없게 하고 결과 셋의 정의를 해제한다.

형식 CLOSE cursor_name;

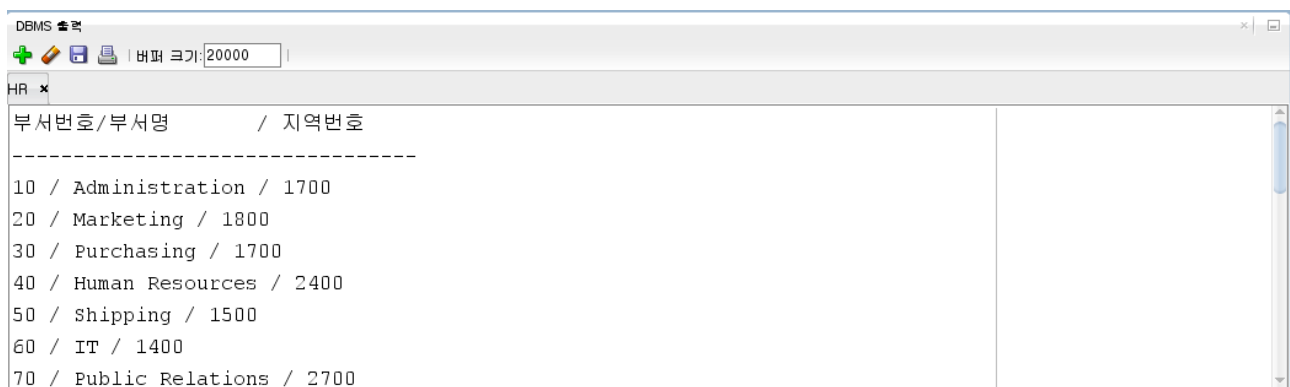
예 CLOSE C1;

```

DECLARE
    VDEPT DEPARTMENTS%ROWTYPE;
    CURSOR C1                -- 커서의 이름
    IS
    SELECT * FROM DEPARTMENTS ; -- 부서 테이블의 전체 내용을 조회한다
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호/  부서명          / 지역번호');
    DBMS_OUTPUT.PUT_LINE('-----');

    OPEN C1;
    --오픈한 C1 커서가 SELECT문에 의해 검색된 한개의 행의 정보를 읽어온다.
    LOOP      --읽어온 정보는 INTO뒤에 모든 컬럼을 기술한다.
        FETCH      C1      INTO      VDEPT.DEPARTMENT_ID,      VDEPT.DEPARTMENT_NAME,
VDEPT.MANAGER_ID, VDEPT.LOCATION_ID;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(VDEPT.DEPARTMENT_ID || ' / ' || VDEPT.DEPARTMENT_NAME || '
/ ' || VDEPT.LOCATION_ID);
        END LOOP;
        CLOSE C1;
END;
/

```



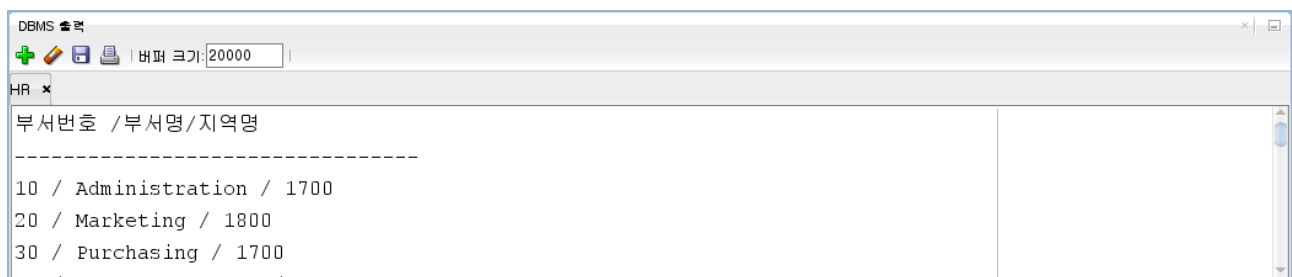
⑨ CURSOR와 FOR LOOP(목시적으로 CURSOR에서 행을 처리한다.)

LOOP에서 각 반복마다 CURSOR를 열고 행을 인출(FETCH)하고 모든 행이 처리되면 자동으로 CURSOR가 CLOSE되므로 사용하기가 편리하다.

형식 **FOR record_name IN cursor_name LOOP**
 statement1;
 statement2;
 END LOOP

```
DECLARE
  VDEPT DEPARTMENTS%ROWTYPE;
  CURSOR C1
  IS
  SELECT * FROM DEPARTMENTS;
BEGIN
  DBMS_OUTPUT.PUT_LINE('부서번호 /부서명/지역명');
  DBMS_OUTPUT.PUT_LINE('-----');
  FOR VDEPT IN C1 LOOP
    DBMS_OUTPUT.PUT_LINE(VDEPT.DEPARTMENT_ID || ' / ' || VDEPT.DEPARTMENT_NAME || '
/ ' || VDEPT.LOCATION_ID);
  END LOOP;
END;
/
```

```
DECLARE
  VDEPT DEPARTMENTS%ROWTYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('부서번호 /부서명/지역명');
  DBMS_OUTPUT.PUT_LINE('-----');
  FOR VDEPT IN (SELECT * FROM DEPARTMENTS) LOOP -- 커서정의 부분을 FOR문에서 직접 사용
    DBMS_OUTPUT.PUT_LINE(VDEPT.DEPARTMENT_ID || ' / ' || VDEPT.DEPARTMENT_NAME ||
' / ' || VDEPT.LOCATION_ID);
  END LOOP;
END;
/
```



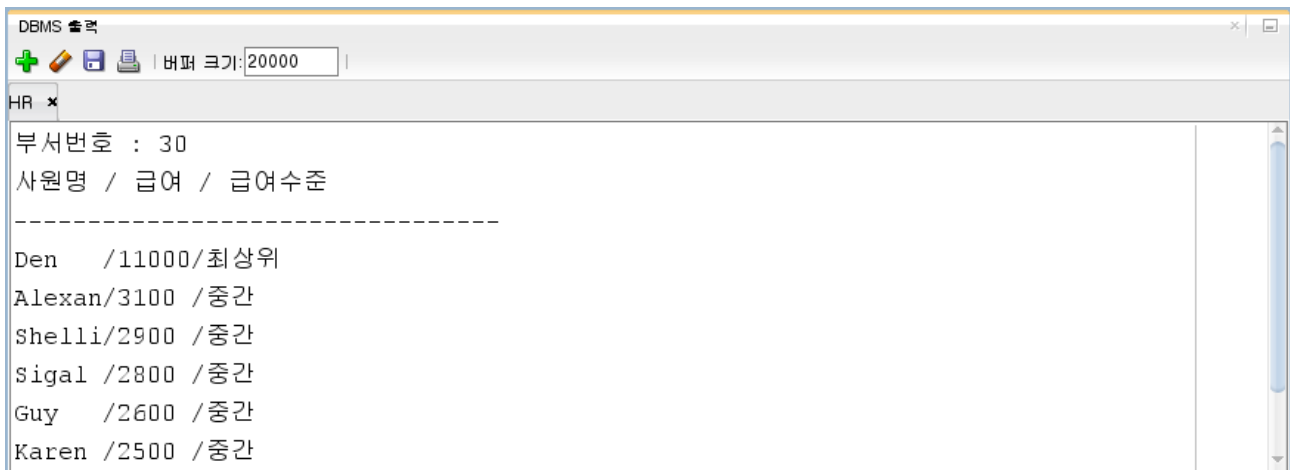
```

-- 부서 번호를 임의의 수로 얻어 레코드를 출력하도록 쿼리를 작성해 주세요.
DECLARE
  VRANDEPTNO EMPLOYEES.DEPARTMENT_ID%TYPE;
  VSALSTR VARCHAR2(20);
  CURSOR CUR_EMP_DEPT(VDEPTNO EMPLOYEES.DEPARTMENT_ID%TYPE)
  IS
  SELECT SALARY, FIRST_NAME FROM EMPLOYEES WHERE DEPARTMENT_ID = VDEPTNO;
BEGIN
  VRANDEPTNO := ROUND(DBMS_RANDOM.VALUE (10, 40), -1);
  DBMS_OUTPUT.PUT_LINE('부서번호 : ' || VRANDEPTNO );
  IF VRANDEPTNO = 40 THEN
    DBMS_OUTPUT.PUT_LINE(VRANDEPTNO || ' 부서에 해당 사원이 없습니다') ;
    RETURN;
  END IF;

  DBMS_OUTPUT.PUT_LINE('사원명 / 급여 / 급여수준');
  DBMS_OUTPUT.PUT_LINE('-----');

  FOR VEMP IN CUR_EMP_DEPT(VRANDEPTNO) LOOP
    IF VEMP.SALARY BETWEEN 1 AND 1500 THEN
      VSALSTR := '낮음';
    ELSIF VEMP.SALARY BETWEEN 1501 AND 5000 THEN
      VSALSTR := '중간';
    ELSIF VEMP.SALARY BETWEEN 4501 AND 8000 THEN
      VSALSTR := '높음';
    ELSE
      VSALSTR := '최상위';
    END IF;
    DBMS_OUTPUT.PUT_LINE(RPAD(VEMP.FIRST_NAME,6) || '/' || RPAD(VEMP.SALARY,5) || '/' ||
VSALSTR);
  END LOOP;
END;
/

```



⑩ 커서변수

한개이상의 쿼리를 연결해 사용할 수 있으며, 변수처럼 커서 변수를 함수나 프로시저의 매개변수로 전달할 수 있다.

• 커서 변수 선언

```
TYPE 커서_타입명 IS REF CURSOR;
커서_변수명 커서_타입명;
```

오라클에서 제공하는 빌트인 커서 타입(결과 집합이 고정되어 있지 않으므로)인 SYS_REFCURSOR이란 타입을 사용하는 것이다. 따라서 SYS_REFCURSOR를 사용할 때는 별도로 커서 타입을 선언할 필요없이 다음과 같이 커서 변수만 선언하면 된다.

```
커서변수 SYS_REFCURSOR;
```

• 커서 변수의 사용

① 커서 변수와 커서 정의 쿼리문 연결

커서를 정의하는 쿼리가 있어야 하는데 커서변수와 쿼리문을 연결할 때 다음과 같이 OPEN...FOR 구문을 사용한다.

```
OPEN 커서변수명 FOR SELECT 문;
```

② 커서 변수에서 결과집합 가져오기

커서를 구성하는 쿼리에 커서 변수까지 연결했으니 커서 변수에 결과 집합을 가져오는 패치 작업이 남았는데, 이때도 FETCH문을 사용한다.

```
FETCH 커서변수명 INTO 변수1, 변수2, ...;
```

• 커서 변수의 사용

```
DECLARE
  VEMPNAME EMPLOYEES.FIRST_NAME%TYPE;
  TYPE EMPCURSOR IS REF CURSOR; -- 커서 타입 선언
  EMPVAR EMPCURSOR; -- 커서 변수 선언
```



```

BEGIN
  -- 커서 변수를 사용한 커서 정의 및 오픈
  OPEN EMPVAR FOR SELECT FIRST_NAME FROM EMPLOYEES WHERE DEPARTMENT_ID = 30;

  -- LOOP문
  LOOP
    -- 커서 변수를 사용해 결과 집합을 EMPNAME 변수에 할당
    FETCH EMPVAR INTO VEMPNAME;
    EXIT WHEN EMPVAR%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(VEMPNAME); -- 사원명을 출력
  END LOOP;
END;
/

```

