


Chapter 11

PL/SQL



데이터베이스 응용

● PL/SQL(Procedural Language extension to SQL)

* 정의

- ☑ **SQL** 언어를 사용한 데이터 조작과 질의문 등을 블록 구조 안에 절차적 단위의 코드로 포함하여 절차적 프로그래밍을 가능하게 한 강력한 트랜잭션 처리 언어

* 특징

- ☑ 오라클 자체에 내장되어 있는 절차적 언어
- ☑ 블록 구조로 되어 있고, **PL/SQL** 자신이 컴파일 엔진을 가짐
- ☑ **SQL**과의 강력한 통합
 - ✍ **DML**, 커서, 트랜잭션 처리, **SQL** 함수 및 연산자 등의 지원
 - ✍ **SQL** 자료형 지원
- ☑ 고성능, 고생산성, 호환성
- ☑ 객체지향 개념 지원 : 추상 자료형(**ADT**)
- ☑ 웹 응용 프로그램 개발 지원
- ☑ 웹 서버 페이지 개발 지원 : **PL/SQL Server Page(PSPs)**

□ PL/SQL

* PL/SQL이 제공하는 기능과 장점

| 기 능 |
|---|
| 블록 내에서 논리적으로 관련된 문장들의 그룹화하여 모듈화된 프로그램 개발 가능 |
| 절차적 언어 구조로 된 프로그램 작성 가능 <ul style="list-style-type: none">- 조건에 따라 일련의 문장을 실행(IF)- 루프에서 반복적으로 일련의 문장을 실행(LOOP)- 명시적 커서(Explicit Cursor)를 이용한 다중 행(Multi-row) 질의 처리 가능 |
| 데이터베이스의 테이블 구조와 컬럼을 기반으로 하는 동적인 변수 선언 가능 |
| 예외(Exception) 처리 루틴을 이용하여 에러 처리, 사용자 정의 에러를 선언하고 예외(Exception) 처리 루틴으로 처리 가능 |
| PL/SQL은 블록 구조로 다수의 SQL 문을 한번에 오라클 데이터베이스로 보내서 처리하므로 응용 프로그램의 성능 향상 가능. |
| PL/SQL은 오라클에 내장되어 있으므로 오라클과 PL/SQL을 지원하는 어떤 호스트에도 프로그램 이식(사용) 가능. |
| 오라클 서버(stored procedure, database trigger, package를 이용)와 오라클 개발 툴의 중간 역할 수행. |

□ PL/SQL

● PL/SQL 프로그램 예제

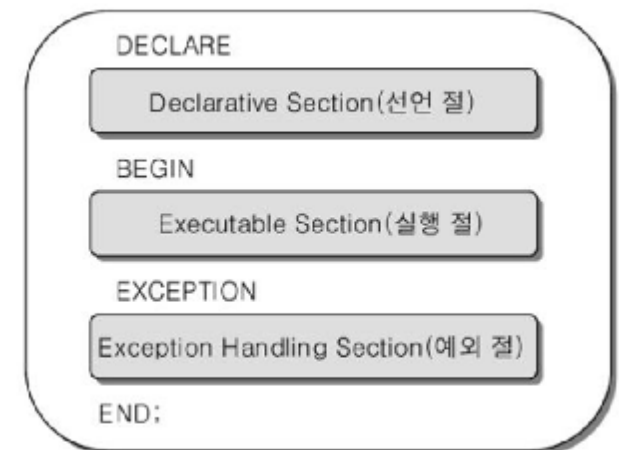
```
PROCEDURE update_part_unitprice (part_id IN INTEGER, new_price IN
NUMBER)
IS
    invalid_part EXCEPTION;
BEGIN
    -- HERE'S AN UPDATE STATEMENT TO UPDATE A DB RECORD
    UPDATE sales.parts
    SET unit_price = new_price
    WHERE id = part_id;
    -- HERE'S AN ERROR-CHECKING STATEMENT
    IF SQL%NOTFOUND THEN
        RAISE invalid_part;
    END IF;
    -- HERE'S AN ERROR-HANDLING ROUTINE
EXCEPTION
    WHEN invalid_part THEN
        raise_application_error(-20000, 'Invalid Part ID');
END update_part_unitprice;
```

□ PL/SQL

● PL/SQL의 블록 구조

※ 선언절(Declaration Section), 실행절(Executable Section), 예외절(Exception Section)로 구성

| 설 명 | 포 함 | |
|-----|--|----|
| 선언부 | 실행절에서 참조할 모든 변수, 상수, 커서(Cursor), 사용자 정의 예외(user - defined exception) 등을 선언 | 옵션 |
| 실행부 | 데이터베이스와 PL/SQL에 있는 데이터 처리를 위한 SQL 문장 포함 | 필수 |
| 예외절 | 에러와 비정상적 조건이 실행부분에서 발생할 때 수행할 내용 명시 | 옵션 |



PL/SQL 블록 구조

□ PL/SQL

```
DECLARE
    v_product_id    product_id.id%TYPE;
BEGIN
    SELECT  id
      INTO  v_product_id
    FROM    product
    WHERE   id = &p_product_id;
    DELETE  FROM    inventory
      WHERE   product_id = v_product_id;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        INSERT INTO exception_table (message)
            VALUES ('예러가 발생했습니다. ');
        COMMIT;
END;
```

PL/SQL 블록의 예

□ PL/SQL

● PL/SQL 프로그램 단위

* 익명 PL/SQL 블록(Anonymous PL/SQL Block)

- ☑ 모든 PL/SQL 환경에서 사용 가능한 이름없는 PL/SQL 블록

* 저장 프로시저(Stored Procedure) 및 함수(Function)

- ☑ 매개 변수를 받을 수 있고 반복해서 사용 가능한 이름있는 PL/SQL 블록

* 패키지(Package)

- ☑ 관련된 프로시저, 함수, 식별자 등을 모두 모은 이름이 있는 PL/SQL 모듈

* 데이터베이스 트리거(Database Triggers)

- ☑ 데이터베이스와 연결되어 자동으로 실행되는 이름있는 PL/SQL 블록

● Stored Procedure와 Function

* 특징

- ☑ SQL 문 또는 다른 PL/SQL 블록으로 구성된 스키마 객체로 데이터베이스에 저장되어 특정 문제를 해결하거나 관련된 작업을 실행하기 위해 사용
- ☑ 매개변수를 사용하여 입력과 출력 값 이용 가능.
- ☑ 프로시저와 함수의 차이
 - ✍ 함수가 항상 하나의 매개변수 값을 되돌려주는 반면 프로시저는 관련 구문 실행

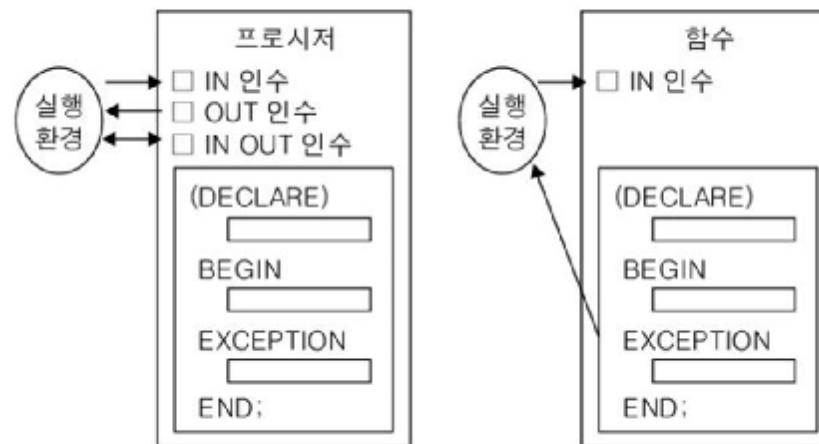
* 프로시저와 함수 실행

- ☑ SQL Developer를 사용(EXECUTE 명령문 실행)
- ☑ 오라클 폼과 같은 데이터베이스 응용 프로그램 코드 안에서 명시적으로 호출
- ☑ 다른 프로시저나 트리거 안에서 명시적으로 호출

□ PL/SQL

● 프로시저와 함수의 차이점

- * 식(expression)의 일부로서 함수를 사용한다.
- * 함수는 값을 반환하는 것이 필수적이다.



| 프로시저 | 함수 |
|-----------------|-------------------|
| PL/SQL 문으로서 실행 | 식의 일부로서 사용 |
| RETURN 문장이 없음 | RETURN 문장 필수 |
| 값을 Return할 수 있음 | 값을 Return하는 것이 필수 |

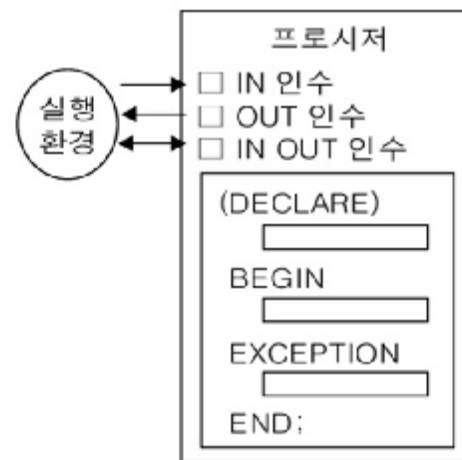
□ PL/SQL

● 매개변수 종류

* 매개변수(Parameter)

- ☑ 실행 환경과 서브 프로시저 사이의 값을 전달하고 받는 역할을 하는 것
- ☑ IN, OUT, IN OUT의 3가지 종류.

| 매개변수 모드 | 설 명 |
|---------|---|
| IN | 실행환경에서 서브 프로시저로 값을 전달 |
| OUT | 프로시저에서 호출한 실행환경으로 값을 전달 |
| IN OUT | IN과 OUT의 기능을 모두 수행한다. 즉, 실행 환경에서 프로시저로 값을 전달하고 프로시저에서 실행환경으로 변경된 값을 전달할 수 있다. |



매개변수의 값의 흐름

□ PL/SQL

매개변수의 특징

| IN | OUT | IN OUT |
|-----------------------------|----------------------------|----------------------------|
| 기본 값, 생략 가능 | 반드시 지정해야 함 | 반드시 지정해야 함 |
| 프로시저에 값 전달 | 프로시저에서 프로시저를 호출한 환경으로 값 변환 | 프로시저에 값을 전달한 후 실행환경으로 값 반환 |
| 상수, 수식 또는 초기화된 변수를 매개변수로 사용 | 초기화되지 않은 변수를 매개변수로 사용 | 초기화된 변수를 매개변수로 사용 |

□ PL/SQL

* 패키지 (Packages)

☑ 특징

- ☞ 관련된 프로시저와 함수들의 그룹으로 커서와 변수들을 함께 공유하여 사용
- ☞ 하나의 단위로 반복해서 사용할 수 있도록 데이터베이스에 함께 저장
- ☞ 패키지 내의 함수와 프로시저는 응용 프로그램이나 사용자에 의해 호출됨.

☑ 구성

- ☞ 명세(Specification)과 몸체(Body) 두 부분으로 구성
 - 명세 부분 : 패키지의 모든 PUBLIC 구성 요소 선언
 - 몸체 부분 : PUBLIC과 PRIVATE 구성 요소들을 정의

☑ 실행

- ☞ 데이터베이스 응용 프로그램에서 패키지 내의 프로시저를 호출하여 사용
- ☞ 작성된 패키지(`employees_management`)에 대한 권한을 가진 사용자가 패키지에 포함된 프로시저를 호출하여 사용
- ☞ 예
 - `hire_employees` 패키지 프로시저 실행 명령어

```
EXECUTE employees_management.hire_employees ('SMITH', 'CLERK', 1037,  
SYSDATE, 500, NULL, 20);
```

□ PL/SQL

● SQL Developer를 이용하여 PL/SQL 블록 작성하기

* 익명(Anonymous) 블록과 단위 프로그램(procedure, function) 작성 방법

- ① SQL 버퍼 내에서 블록을 정의한 다음 버퍼의 내용을 실행
- ② SQL Developer 스크립트 파일의 부분으로 블록을 정의하여 스크립트 파일을 실행

* SQL Developer에서 PL/SQL 블록을 작성하거나 실행하는 데 필요한 명령

| SQL*Plus 명령 | 설 명 |
|-------------|---------------------------------------|
| ACCEPT | 사용자의 입력(값)을 읽어 변수에 저장 |
| VARIABLE | 앞에 콜론(:)을 써서 PL/SQL에서 참조할 수 있는 변수를 선언 |
| PRINT | 변수의 현재 값을 표시 |
| EXECUTE | 하나의 PL/SQL 문을 실행 |

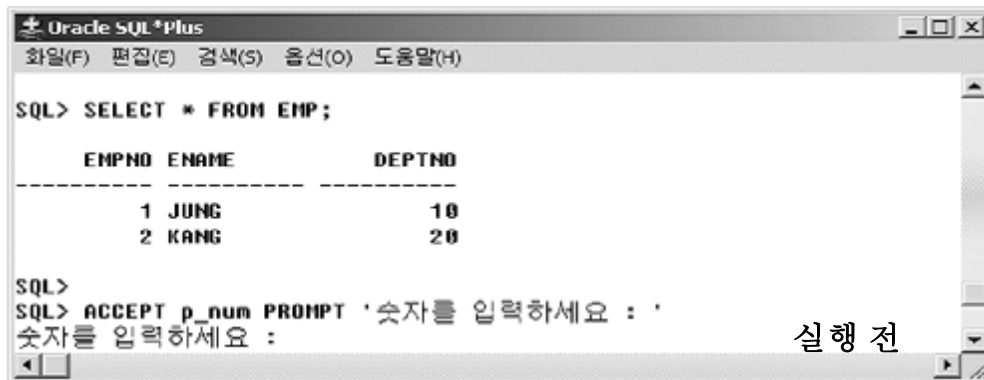
□ PL/SQL

* 익명 PL/SQL 블록 작성 및 실행하기

☑ 예

✍ **ACCEPT** 문을 사용하여 사용자로부터 값을 입력 받아 짝수이면 **DEPTNO**를 변경하고, 홀수이면 행을 삽입하는 구문

```
SQL> ACCEPT p_num PROMPT '숫자를 입력하세요 : '  
숫자를 입력하세요 : 20  
SQL> DECLARE  
    v_num NUMBER := &p_num;  
BEGIN  
    IF MOD(v_num, 2) = 0 THEN  
        UPDATE EMP  
        SET DEPTNO = '14' ;  
  
    ELSE  
        INSERT INTO EMP VALUES (3, 'PARK', 30);  
  
    END IF;  
END;  
/
```



Oracle SQL*Plus

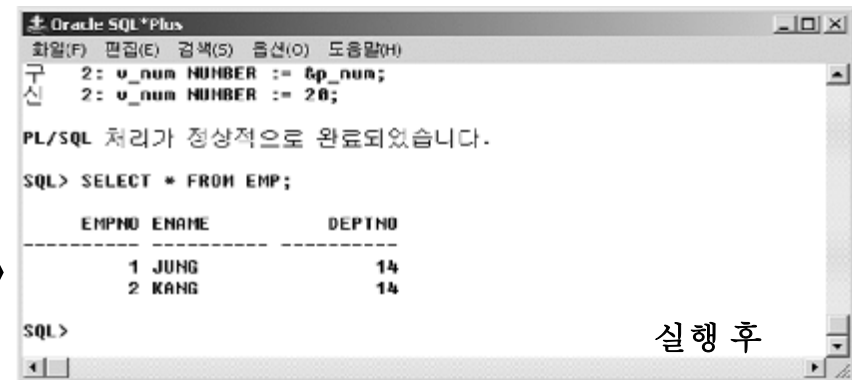
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

```
SQL> SELECT * FROM EMP;
```

| EMPNO | ENAME | DEPTNO |
|-------|-------|--------|
| 1 | JUNG | 10 |
| 2 | KANG | 20 |

```
SQL>  
SQL> ACCEPT p_num PROMPT '숫자를 입력하세요 : '  
숫자를 입력하세요 : 
```

실행 전



Oracle SQL*Plus

화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

```
구 2: v_num NUMBER := &p_num;  
신 2: v_num NUMBER := 20;  
  
PL/SQL 처리가 정상적으로 완료되었습니다.  
SQL> SELECT * FROM EMP;
```

| EMPNO | ENAME | DEPTNO |
|-------|-------|--------|
| 1 | JUNG | 14 |
| 2 | KANG | 14 |

```
SQL>
```

실행 후

□ PL/SQL

● 저장 프로시저 작성 및 실행하기

- * **CREATE PROCEDURE**나 **CREATE OR REPLACE** 구문을 사용하여 생성하며 프로시저의 **PL/SQL** 블록은 **IS**로 시작
- * 저장 프로시저를 삭제는 **DROP PROCEDURE** 문 사용

```
CREATE PROCEDURE 프로시저이름  
    ( 매개변수이름 [ IN | OUT | IN OUT ] 데이터타입  
      [= DEFAULT ] 식 ], ... )  
  
IS  
    [ DECLARE ]  
        --  
BEGIN  
        --  
    [ EXCEPTION ]  
        --  
END 프로시저이름;
```

□ PL/SQL

```
CREATE PROCEDURE park_salary
( v_emp_no      IN  NUMBER,
  v_new_salary  IN  NUMBER )
IS
BEGIN
    UPDATE SALARY
    SET      SALARY = v_new_salary
    WHERE   EMP_ID = v_emp_no ;
    COMMIT;
END park_salary;
```

Oracle SQL*Plus

파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> SELECT * FROM SALARY;

| EMP_ID | EMP_NAME | JOB | DEPT_NO | SALARY |
|--------|----------|----------|---------|---------|
| 1001 | KIM | ENGINEER | 10 | 3500000 |
| 1002 | PARK | MANAGER | 11 | 4000000 |
| 1003 | LEE | SALES | 12 | 2400000 |

실행 전

Oracle SQL*Plus

파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

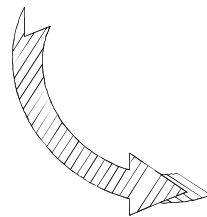
SQL> EXECUTE park_salary (1002, 2500000)

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>

SQL> |

프로시저 실행



Oracle SQL*Plus

파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL>

SQL> SELECT * FROM SALARY;

| EMP_ID | EMP_NAME | JOB | DEPT_NO | SALARY |
|--------|----------|----------|---------|---------|
| 1001 | KIM | ENGINEER | 10 | 3500000 |
| 1002 | PARK | MANAGER | 11 | 2500000 |
| 1003 | LEE | SALES | 12 | 2400000 |

실행 후

SQL> |

□ PL/SQL

● 함수(Function)

* 특징

- ☑ CREATE (OR REPLACE) FUNCTION 문으로 생성
- ☑ PL/SQL 블록은 함수가 수행할 내용을 정의하는 부분으로, 적어도 한 개의 RETURN 문이 있어야 함.

* 생성 구문

```
CREATE OR REPLACE FUNCTION 함수이름  
    [(argument,...)]  
    RETURN 반환되는 값의 데이터 타입  
  
IS  
    [ 변수 선언 ]  
BEGIN  
    pl/sql_block  
END;
```

□ PL/SQL

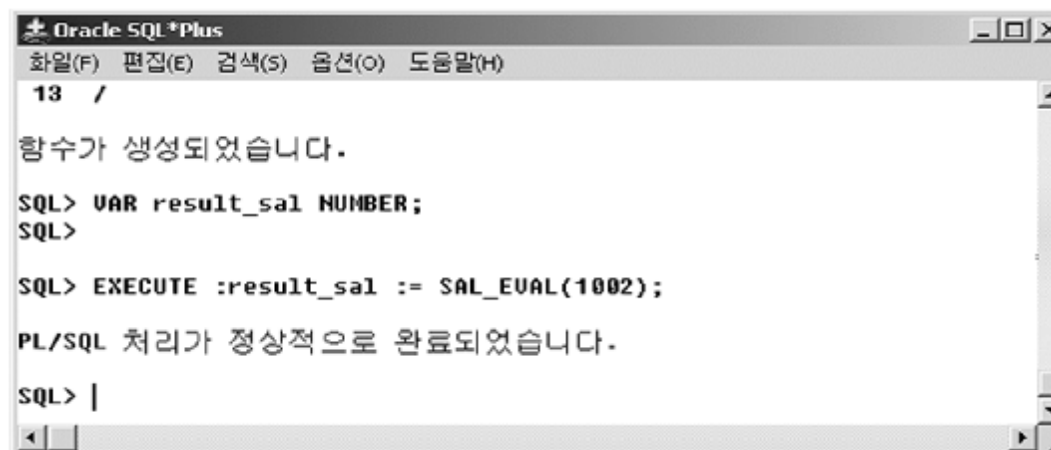
☑ 예

```
FUNCTION sal_eval ( v_emp_no NUMBER )  
    RETURN NUMBER  
IS  
    v_salary    NUMBER;  
BEGIN  
    SELECT salary INTO v_salary FROM salary  
        WHERE v_emp_no = emp_no;  
    RETURN ( v_salary );  
END sal_eval;
```

Sal_eval 함수 선언

```
VAR result_sal NUMBER
```

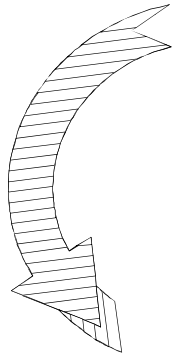
함수로부터 반환되는 값을 저장할 변수를 선언



```
Oracle SQL*Plus  
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)  
13 /  
함수가 생성되었습니다.  
SQL> VAR result_sal NUMBER;  
SQL>  
SQL> EXECUTE :result_sal := SAL_EVAL(1002);  
PL/SQL 처리가 정상적으로 완료되었습니다.  
SQL> |
```

함수 실행하기

□ PL/SQL



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL>
SQL> PRINT result_sal

RESULT_SAL
-----
      2500000

SQL> |
```

선언된 변수의 값을 출력

□ PL/SQL

● 식별자

* 정의

- ☑ 변수, 커서, 타입 등의 **PL/SQL** 객체의 이름에 사용

* 특징

- ☑ 식별자의 첫 번째 문자는 알파벳으로 시작
 - ✎ 이후로 알파벳, 숫자, \$, #, _ 등을 사용
 - ✎ 최대 길이는 **30**바이트 이내
- ☑ 대소문자를 구별하지 않는다.
- ☑ 예약어(**Reserved Word**)를 식별자로 사용할 수 없다.
- ☑ 따옴표(“”)를 사용한 식별자
 - ✎ 대소문자를 구별
 - ✎ 위 식별자 작성 규칙을 만족안해도 됨

□ PL/SQL

● 식별자 예제

```
"HELLO" varchar2(10) := 'hello';  
...  
DBMS_Output.Put_Line("HELLO");
```

```
"BEGIN" varchar2(15) := 'UPPERCASE';  
"Begin" varchar2(15) := 'Initial Capital';  
"begin" varchar2(15) := 'lowercase';  
...
```

```
DBMS_Output.Put_Line("BEGIN");  
DBMS_Output.Put_Line("Begin");  
DBMS_Output.Put_Line("begin");
```

□ PL/SQL

● 연산자

※ 논리, 산술 및 연결 연산자와 지수 연산자(**) 사용 가능

| 종류 | 연산자 | 설명 |
|-------|----------------|--------------------|
| 일반연산자 | +, -, *, / | 덧셈, 뺄셈, 곱셈, 나눗셈 |
| | =, <, > | 관계 연산자 |
| | (,) | 설명 또는 리스트 구분자 |
| | ; | 문장 끝마침 구분자 |
| | % | 속성 인자 |
| | , | 아이템 또는 문자열 구분자 |
| 단수연산자 | . | 컴포넌트 선택 |
| | @ | 원격 접근 지시자 |
| | : | 호스트 변수 지시자 |
| 복수연산자 | ** | 지수 연산자 |
| | <>, <=, >=, != | 관계 연산자 |
| | := | 할당(assignment) 연산자 |
| | .. | 범위 연산자 |
| | | 문자열 접합 연산자 |
| | <<, >> | label 연산자 |
| | -- | 주석 연산자 : 한 행 이하 |
| | /* */ | 주석 연산자 : 복수 행인 경우 |

□ PL/SQL

● 주석

* 종류

- ☑ 단수 주석과 복수 주석

* 단수 주석

- ☑ 두 개의 대시(--)를 사용하여 한 행만을 주석으로 정의하기 위해 사용

```
FUNCTION sal_evd ( v_emp_no NUMBER )  
    RETURN NUMBER  
IS  
    v_salary    NUMBER;  
BEGIN -- PL/SQL 블록 시작  
    SELECT salary INTO v_salary FROM salary  
        WHERE v_emp_no = emp_no;  
    RETURN ( v_salary );  
END sal_evd; -- PL/SQL 블록 끝
```

□ PL/SQL

* 복수 주석

- ☑ 복수 주석은 두 줄 이상의 행에 주석을 달 때 사용하며 /* 로 시작하여 */로 끝난다.

```
FUNCTION sal_evl ( v_emp_no NUMBER )  
    RETURN NUMBER  
IS  
    v_salary    NUMBER;  
BEGIN  
    /* 이 기호는 한 줄 이상의 주석을 작성하는 경우  
    사용하는 기호이다. */  
    SELECT salary INTO v_salary FROM salary  
        WHERE v_emp_no = emp_no;  
    RETURN ( v_salary );  
END sal_evl;
```


● 변수와 데이터 타입

* 변수 선언하기

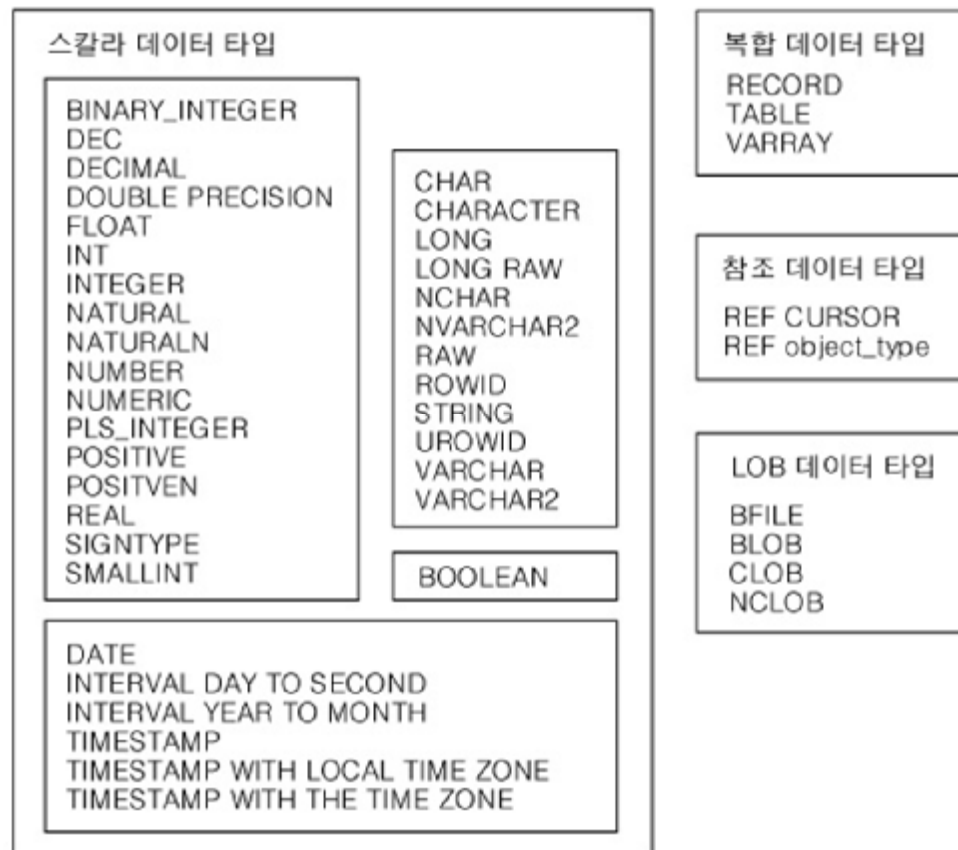
☑ 반환되는 값을 저장하기 위한 변수 선언

식별자 [CONSTANT] 데이터 타입 [NOT NULL] [:= 상수 값 또는 식] ;

- ✎ 식별자의 이름은 **SQL**에서 객체의 이름을 작성하는 규칙을 따른다.
- ✎ 식별자를 상수로 지정하고 싶은 경우는 **CONSTANT**라는 키워드를 명시하고 반드시 초기 값을 할당한다.
- ✎ **NOT NULL**이 정의되어 있으면 초기 값을 반드시 지정하고, 정의되어 있지 않을 때는 생략 가능하다.
- ✎ 초기 값을 할당 연산자(:=)를 사용하여 정의한다.
- ✎ 초기 값을 정의하지 않으면 식별자는 **NULL** 값을 가지게 된다.
- ✎ 일반적으로 한 줄에 한 개의 식별자를 정의한다.

□ PL/SQL

● 미리정의된 데이터 타입



□ PL/SQL

● 스칼라 데이터 타입

| 데이터 타입 | 설명 |
|-------------------------------|--|
| BINARY_INTEGER PLS_INTEGER | -2147483647~ 2147483647 사이의 정수, 기본 값=1 BINARY_INTEGER의 서브 타입 : NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE - 작은 저장공간, H/W 연산 사용(효율적) |
| NUMBER[(Precision, Scale)] | 고정 및 부동 소숫점에 대한 기본 유형(정수 및 실수) NUMBER의 서브 타입 : DEC, DECIMAL DOUBLEPRECISION, FLOAT, INTEGER, INT NUMERIC, REAL, SMALLINT |
| CHAR[(최대길이)] | 32767바이트의 고정 길이 문자. 기본 값은 1 |
| LONG | 32760바이트까지의 가변 길이 문자 |
| VARCHAR2(최대길이) | 32767Byte까지의 가변 길이 문자 데이터 varchar2의 서브 타입 : STRING, VARCHAR |
| DATE | 날짜와 시간에 대한 기본 형 |
| BOOLEAN | 논리연산에 사용되는 세 가지 값(TRUE, FALSE, NULL)을 저장 |

* 초기 값을 지정하는 경우 := 사용

```
v_price CONTANT NUMBER(4,2) := 12.34 ;    -- 상수 숫자 선언
v_name VARCHAR2(20) ;
v_Bir_Type CHAR(1) ;
v_flag BOOLEAN NOT NULL := TRUE ;
v_birthday DATE;
```

스칼라 데이터 타입 선언 예제

□ PL/SQL

● %TYPE

- * 이미 선언된 다른 변수나 데이터베이스 컬럼의 데이터 타입을 이용하여 선언
- * **%TYPE** 앞에는 데이터베이스 테이블과 컬럼 그리고 이미 선언한 변수이름이 올 수 있다.
- * 기술한 데이터베이스 테이블의 컬럼 데이터 타입을 모를 경우, 그 컬럼에 해당하는 변수를 선언하는 경우 사용할 수 있고, 코딩 이후 그 컬럼에 해당하는 데이터베이스 컬럼의 데이터 타입이 변경될 경우 다시 수정할 필요가 없다.
- * 초기 값 지정 가능
- * 예

```
v_emp_no emp.empno%TYPE := 1000;  
v_emp_name emp.ename%TYPE ;
```

□ PL/SQL

```
create or replace PROCEDURE Print_Movie_Info
(tt IN Movie.title%TYPE, yy IN Movie.year%TYPE)
IS
    std_name    Movie.studioName%TYPE;
    prod_name    MovieExec.Name%TYPE;
BEGIN
    DBMS_OUTPUT.ENABLE;
    SELECT studioname, ex.name
    INTO std_name, prod_name
    FROM Movie, MovieExec ex
    WHERE producerNo = certNo and title = tt and year = yy;
    -- 결과 출력
    DBMS_OUTPUT.PUT_LINE('영화 : '||tt||'('||yy||)');
    DBMS_OUTPUT.PUT_LINE('  영화사 이름 : '||std_name);
    DBMS_OUTPUT.PUT_LINE('  제작자 이름 : '||prod_name);

END;
```

```
SQL> set serveroutput on;
SQL> execute print_movie_info('star wars', 1977);
영화 : star wars(1977)
영화사 이름 : fox
제작자 이름 : george lucas
```

PL/SQL 처리가 정상적으로 완료되었습니다.

□ PL/SQL

● 복합 데이터 타입

* 특징

- ☑ 하나 이상의 데이터 값을 갖는 데이터 타입으로 배열과 같은 역할을 함
- ☑ **PL/SQL** 테이블과 레코드, **%ROWTYPE**이 복합 데이터 타입에 포함

* %ROWTYPE

- ☑ **%ROWTYPE** 앞에는 데이터베이스 테이블의 이름이 위치
- ☑ 지정된 테이블의 구조와 동일한 구조를 갖는 변수 선언 가능.
- ☑ 데이터베이스 컬럼의 수나 데이터 타입을 알지 못할 때 사용

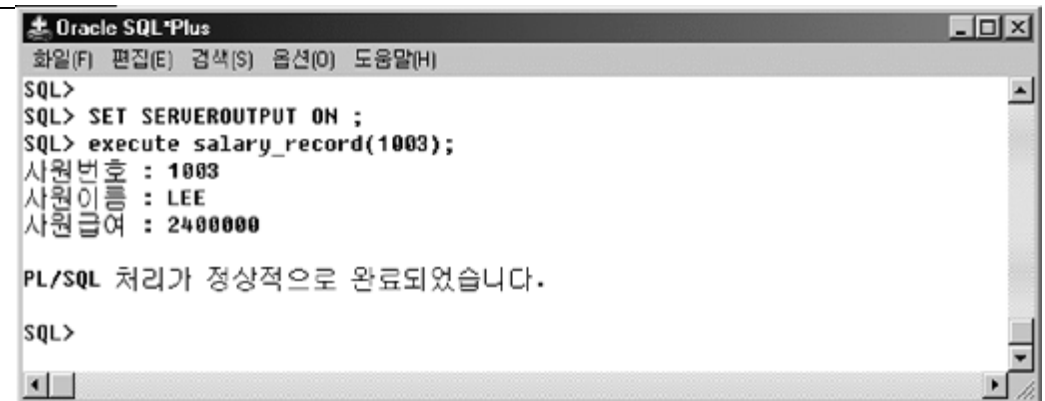
```
SQL> CREATE OR REPLACE PROCEDURE Salary_record
      (p_emp_no IN SALARY.EMP_NO%TYPE)
IS
  sal_record    SALARY%ROWTYPE;

BEGIN
  DBMS_OUTPUT.ENABLE;
  SELECT * INTO sal_record
  FROM SALARY
  WHERE EMP_NO = 1003 ;
```

-- 결과 값 출력

```
DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || sal_record.emp_no );
DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || sal_record.emp_name );
DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || sal_record.sal );

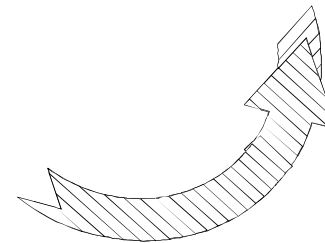
END;  /
```



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> SET SERVEROUTPUT ON ;
SQL> execute salary_record(1003);
사원번호 : 1003
사원이름 : LEE
사원급여 : 2400000

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>
```



□ *PL/SQL*

```
create or replace PROCEDURE Get_Studio_Name  
    (mtitle IN movie.title%TYPE, myear IN movie.year%TYPE,  
     sName OUT movie.studioname%TYPE)
```

```
IS
```

```
    res  Movie%ROWTYPE;
```

```
BEGIN
```

```
    SELECT *
```

```
    INTO res
```

```
    FROM Movie
```

```
    WHERE title = mtitle AND year = myear;
```

```
    sName := res.studioname;
```

```
EXCEPTION
```

```
    WHEN OTHERS THEN
```

```
        sName := NULL;
```

```
END;
```

```
var num number;
```

```
var stdname varchar2(30);
```

```
execute get_studio_name('star wars', 1977, :stdname);
```

```
print stdname;
```

□ PL/SQL

● PL/SQL의 레코드

* 특징

- ☑ 사용자 정의 레코드
- ☑ 여러 개의 데이터 타입을 갖는 변수들의 집합으로, 스칼라, 레코드, PL/SQL 테이블 데이터 타입 중 하나 이상의 요소로 구성
- ☑ PL/SQL의 레코드는 논리적 단위로서 필드 집합 처리
- ☑ PL/SQL 테이블과 다르게 개별 필드의 이름 부여 가능
- ☑ 선언할 때 초기화가 가능
- ☑ PL/SQL 블록에서 테이블의 행을 읽어올 때 편리하게 사용.

* PL/SQL 레코드를 선언하는 문법

```
TYPE 유형이름 IS RECORD
    (필드이름_1 필드 유형 [ NOT NULL {:= | DEFAULT} 식 ],
    (필드이름_2 필드 유형 [ NOT NULL {:= | DEFAULT} 식], ..);
식별자 유형이름;
```


□ PL/SQL

● 레코드 사용 예제

```
TYPE part_record IS RECORD (  
    id sales.parts.id%TYPE,  
    unit_price sales.parts.unit_price%TYPE,  
    description sales.parts.description%TYPE  
);  
current_part part_record;  
  
TYPE parts_table IS TABLE OF sales.parts%ROWTYPE;  
current_table parts_table;
```

□ PL/SQL

● 레코드 필드를 가지는 레코드 예제

```
DECLARE
  TYPE name_rec IS RECORD (
    first employees.first_name%TYPE,
    last employees.last_name%TYPE
  );
  TYPE contact IS RECORD (
    name name_rec,           -- nested record
    phone employees.phone_number%TYPE
  );
  friend contact;
BEGIN
  friend.name.first := 'John';
  friend.name.last := 'Smith';
  friend.phone := '1-650-555-1234';
  DBMS_OUTPUT.PUT_LINE (
    friend.name.first || ' ' || friend.name.last || ', ' || friend.phone );
END;
```

□ PL/SQL

```
CREATE OR REPLACE PROCEDURE PrintPresident(limit IN NUMBER)
IS
    i NUMBER;
    res MovieEXEC%ROWTYPE;
BEGIN
    DBMS_OUTPUT.ENABLE;
    FOR i IN 1..limit LOOP
        BEGIN
            SELECT * into res
            FROM MovieExec
            WHERE i = certno;
            DBMS_OUTPUT.PUT_LINE(i);
            DBMS_OUTPUT.PUT_LINE('제 작 자 : '||res.Name);
            DBMS_OUTPUT.PUT_LINE(' 제 작 자 번 호 : '||res.certNo);
            DBMS_OUTPUT.PUT_LINE(' 제 작 자 재 산 : '||res.netWorth);
        EXCEPTION
            WHEN OTHERS THEN
                NULL;
        END;
    END LOOP;
END;
```

□ *PL/SQL*

```
CREATE OR REPLACE FUNCTION get_length  
  (movietitle IN movie.title%type, movieyear movie.year%type)  
RETURN INTEGER  
IS  
  return_length INTEGER;  
BEGIN  
  SELECT length into return_length FROM system.movie  
  WHERE lower(title) = lower(movietitle) AND movieyear = year  
  RETURN return_length;  
EXCEPTION  
  WHEN OTHERS THEN  
    RETURN -1;  
END get_length ;
```

□ *PL/SQL*

```
CREATE OR REPLACE FUNCTION GetProdName
  (tt IN Movie.title%TYPE, yy IN Movie.year%TYPE)
RETURN  MovieExec.NAME%TYPE
IS
  prod_name      MovieExec.Name%TYPE;
BEGIN
  SELECT name
  INTO prod_name
  FROM Movie, MovieExec
  WHERE producerNo = certNo AND title = tt AND year = yy;

  RETURN prod_name;
END;
```

□ PL/SQL

● Associative Array(이전 명칭은 PL/SQL 테이블, index-by 테이블)

- * C 언어의 일차원 배열과 유사하며 데이터베이스 테이블을 참조할 수 있다.
 - ☑ 같은 타입의 요소들의 집합체이고 지정된 타입의 인덱스로 각 요소의 접근이 가능
- * 메모리 공간 차지
- * 디스크 공간 요구하지 않으므로 **DML** 문장 사용 불가
 - ☑ **collection method**를 사용
- * 생성 문법
 - ☑ 테이블 데이터 타입을 선언한 후 그 데이터 타입으로 변수를 선언하여 사용.

```
TYPE type_name IS TABLE OF element_type [NOT NULL]
    INDEX BY [BINARY_INTEGER | PLS_INTEGER | VARCHAR2(size_limit)];
```

□ *PL/SQL*

DECLARE

-- Associative array indexed by string:

TYPE population IS TABLE OF NUMBER -- Associative array type
INDEX BY VARCHAR2(64); -- indexed by string

city_population **population**; -- Associative array variable

i VARCHAR2(64); -- Scalar variable

BEGIN

-- Add elements (key-value pairs) to associative array:

city_population('Smallville') := 2000;

city_population('Midland') := 750000;

city_population('Megalopolis') := 1000000;

-- Change value associated with key 'Smallville':

city_population('Smallville') := 2001;

-- Print associative array:

i := city_population.FIRST; -- Get first element of array

WHILE i IS NOT NULL LOOP

DBMS_Output.PUT_LINE('Population of ' || i || ' is ' || city_population(i));

i := **city_population.NEXT(i)**; -- Get next element of array

END LOOP;

END;

□ PL/SQL

-- Using Cursor.sql

DECLARE

TYPE MovieTY IS TABLE OF movie%rowtype INDEX BY VARCHAR2(255);

MovieArray MovieTY;

m Movie%ROWTYPE;

i VARCHAR2(255);

CURSOR c1 IS SELECT * FROM Movie;

BEGIN

OPEN c1;

LOOP

FETCH c1 INTO m;

EXIT WHEN c1%NOTFOUND;

MovieArray(m.title||', '||m.year) := m;

END LOOP;

CLOSE c1;

i := MovieArray.FIRST;

WHILE i IS NOT NULL LOOP

DBMS_OUTPUT.PUT_LINE([' || i || ' ' || Upper(MovieArray(i).studioName));

i := MovieArray.NEXT(i);

END LOOP;

END;

실행결과

[Chicago,2002] DISNEY

[aliens,1986] FOX

[aliens,1996] FOX

[blade runner,1982] WARNER BROS

[coal miner's daughter,1980] MGM

[fool's running,2000] MGM

[get shorty,1995] MGM

[getaway,1994] FOX

...

□ PL/SQL

● Varrays(Variable-Size Arrays)

- * 1~MAXSIZE 까지의 요소들로 이루어진 배열
- * 1~MAXSIZE 사이의 첨자값으로 각 요소 접근
- * MAXSIZE 까지의 공간을 계속 확보
- * 빈 varray가 되거나 임의의 값으로 초기화되어야 함.

● Varray Grades의 구조

- * MAXSIZE = 10, 현재 크기 = 7
- * Grades(n) = n번째 Grades 요소

Varray Grades

| | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|--|--|--|
| B | C | A | A | C | D | B | | | |
| (1) | (2) | (3) | (4) | (5) | (6) | (7) | | | |

Maximum Size = 10

□ PL/SQL

● Varrays의 예제

```
TYPE Foursome IS VARRAY(4) OF VARCHAR2(15); -- VARRAY type
TYPE parts_varray IS VARRAY(3) OF Movie.title%TYPE;
team Foursome := Foursome('John', 'Mary', 'Alberto', 'Juanita');
current parts_varray := parts_varray(); -- EMPTY
team(3) := 'Pierre'; -- Change values of two elements
team(4) := 'Yvonne';
current.EXTEND(4); -- Append spaces for 4 elements
current(1) := 'KS';
```

□ PL/SQL

● Varray 예제

```
DECLARE
    TYPE population IS VARRAY(3) OF NUMBER;
    city_population population := population();    -- Initialize by empty
    i NUMBER;    -- Scalar variable
BEGIN
    city_population.EXTEND(2); -- append 2 element spaces
    i := city_population.FIRST; -- Get first element of array
    WHILE i IS NOT NULL LOOP
        DBMS_Output.PUT_LINE('Population of ' || i || ' is ' || city_population(i));
        i := city_population.NEXT(i); -- Get next element of array
    END LOOP;
    -- append 1 element space
    city_population.EXTEND(1);
    DBMS_OUTPUT.PUT_LINE('After Extend ...');
    city_population := population(100, 200, 300);
    i := city_population.FIRST; -- Get first element of array
    WHILE i IS NOT NULL LOOP
        DBMS_Output.PUT_LINE('Population of ' || i || ' is ' || city_population(i));
        i := city_population.NEXT(i); -- Get next element of array
    END LOOP;
END;
```

실행결과

```
Population of 1 is
Population of 2 is
After Extend ...
Population of 1 is 100
Population of 2 is 200
Population of 3 is 300
```

□ PL/SQL

● Nested Tables

- * Varray와 유사하나 요소들의 최대 수가 없음
- * 빈 **nested table**이 되거나 임의의 값으로 초기화가 되어야 함
- * 요소들이 차지하는 공간이 동적으로 변화 됨
 - ☑ 요소를 삭제시 해당 공간이 삭제 됨

● 예 제

```
TYPE Roster IS TABLE OF VARCHAR2(15); -- nested table type
names Roster := Roster('D Caruso', 'J Hamil', 'D Piro', 'R Singh');
names(3) := 'P Perez'; -- Change value of one element
FOR i IN names.FIRST .. names.LAST LOOP -- For first to last element
    DBMS_OUTPUT.PUT_LINE(names(i));
END LOOP;
```

Array of Integers

| | | | | | | | | | |
|------|------|------|------|------|------|------|------|------|-------|
| 321 | 17 | 99 | 407 | 83 | 622 | 105 | 19 | 67 | 278 |
| x(1) | x(2) | x(3) | x(4) | x(5) | x(6) | x(7) | x(8) | x(9) | x(10) |

Fixed Upper Bound

Nested Table after Deletions

| | | | | | | | | | |
|------|--|------|------|--|------|------|------|--|-------|
| 321 | | 99 | 407 | | 622 | 105 | 19 | | 278 |
| x(1) | | x(3) | x(4) | | x(6) | x(7) | x(8) | | x(10) |

Unbounded →

□ PL/SQL

```
DECLARE
    TYPE population IS TABLE OF VARCHAR2(50);
    city_population population := population();
    i NUMBER;                -- Scalar variable
BEGIN
    city_population.EXTEND(3);
    /*
    city_Population(1) := 'First';
    city_Population(2) := 'Second';
    city_Population(3) := 'Third';
    */
    city_Population := population('First', 'Second', 'Third');
    i := city_population.FIRST; -- Get first element of nested table
    WHILE i IS NOT NULL LOOP
        DBMS_Output.PUT_LINE('Population of ' || i || ' is ' || city_population(i));
        i := city_population.NEXT(i); -- Get next element of nested table
    END LOOP;

    city_population.EXTEND(2); -- Append 2 spaces
    city_population(4) := 'Fourth';
    city_population(5) := 'Fifth';
    DBMS_OUTPUT.PUT_LINE('After Extend ...');
    i := city_population.FIRST; -- Get first element of nested table
    WHILE i IS NOT NULL LOOP
        DBMS_Output.PUT_LINE('Population of ' || i || ' is ' || city_population(i));
        i := city_population.NEXT(i); -- Get next element of nested table
    END LOOP;
END;
```

실행결과

Population of 1 is First
Population of 2 is Second
Population of 3 is Third
After Extend ...
Population of 1 is First
Population of 2 is Second
Population of 3 is Third
Population of 4 is Fourth
Population of 5 is Fifth

● Nested Table과 Varray를 위한 Collection Method

- * **EXISTS(x)** : x번째 요소가 Nested Table 또는 Varray에 존재하면 **TRUE**, 그렇지 않으면 **FALSE**
- * **COUNT** : 요소들의 수
- * **LIMIT** : Varray에 대해서 최대 저장할 수 있는 요소의 수
- * **FIRST/LAST** : Nested Table 또는 Varray의 첫번째/마지막 요소의 첨자값
- * **PRIOR(x)/NEXT(x)** : Nested Table 또는 Varray에서 x번째 요소 이전/이후 요소의 첨자값
- * **EXTEND(x,y)** : Nested Table 또는 Varray의 y번째 요소를 x개 복사해서 추가
 - ☑ **EXTEND(x)** : x개의 요소 공간을 추가 (varray의 초기 선언된 크기 초과 불가)
- * **TRIM(x)** : Nested Table 또는 Varray의 마지막에서 x개의 요소를 삭제 – *Associative array*에는 사용 불가
- * **DELETE(x,y)** : Nested Table 또는 Associative Array의 x에서 y번째 요소들을 삭제 – *Varray*에는 사용 불가
- * **EXTEND(x)** : Nested Table 또는 Varray에 x개의 빈 요소들을 추가

□ PL/SQL

```
record_count := current_parts_table.COUNT;  
current_parts_record := current_parts_table.FIRST;  
current_parts_table.DELETE(3); -- 3번째 요소 삭제  
current_parts_table.DELETE(3,6); -- 3~6번째 요소 삭제  
current_parts_table.DELETE(6,3); -- 무시 됨  
current_parts_table.DELETE; -- 모든 요소 삭제  
current_parts_record := current_parts_table.PRIOR(current_parts_table.FIRST);  
current_parts_table.EXTEND(3,6); -- 6번째 요소 3개를 추가  
current_parts_table.EXTEND; -- 1개 빈 요소 추가  
current_parts_table.TRIM(3); -- 마지막 3개 요소 삭제
```

```
FOR i IN courses.FIRST..courses.LAST LOOP ...
```

□ PL/SQL

DECLARE

TYPE CourseList IS TABLE OF VARCHAR2(10);

courses CourseList;

BEGIN

courses := CourseList('Biol 4412', 'Psyc 3112', 'Anth 3001');

courses.DELETE(courses.LAST); -- delete 3rd element

courses.TRIM(courses.COUNT);

DBMS_OUTPUT.PUT_LINE(courses(1)); -- prints 'Biol 4412'

END;

- **Nested Table**에서 **DELETE**된 공간은 실제로 삭제되지만 해당 항목에 대한 정보는 유지됨

- **TRIM**은 **DELETE**된 공간까지 포함해서 항목을 삭제

● **collection** 관련 예외처리 코드

DECLARE

TYPE NumList IS TABLE OF NUMBER;

nums NumList; -- NULL 상태임

BEGIN

nums(1) := 1; -- raises **COLLECTION_IS_NULL**

nums := NumList(1,2); -- initialize table

nums(NULL) := 3 -- raises **VALUE_ERROR**

nums(0) := 3; -- raises **SUBSCRIPT_OUTSIDE_LIMIT**

nums(3) := 3; -- raises **SUBSCRIPT_BEYOND_COUNT**

nums.DELETE(1); -- delete element 1

IF nums(1) = 1 THEN ... -- raises **NO_DATA_FOUND**

□ PL/SQL

● multi-level collections

```
declare
    type movie_type is table of movie%rowtype index by pls_integer;
    movie_tab movie_type;
    rec      movie%rowtype;
    cursor cmovie is select * from movie;
begin
    open cmovie;
    loop
        fetch cmovie into rec;
        exit when cmovie%NOTFOUND;
        movie_tab(cmovie%ROWCOUNT) := rec;
    end loop;
    for i in movie_tab.first .. movie_tab.last loop
        dbms_output.put_line(movie_tab(i).title || '(' || movie_tab(i).year ||)');
    end loop;
end;
```

● 오라클 PL/SQL에서 사용 가능한 SQL 문

* DML 문

☑ SELECT, INSERT, UPDATE, DELETE, SET TRANSACTION
문

* 트랜잭션 제어 명령어

☑ COMMIT, ROLLBACK, SAVEPOINT

* 함수

* 의사 컬럼(Pseudocolumns)

* 연산자

□ PL/SQL

● SELECT 문 사용하기

* PL/SQL 내에서 **SELECT** 문

- ☑ 데이터를 추출하기 위해 사용
- ☑ 사용 구문

```
SELECT select_list INTO 변수이름 | 레코드이름  
FROM 테이블이름  
WHERE 조건 ;
```

☑ 주의 사항

- ✎ **SELECT** 문은 반드시 하나의 데이터 행만을 추출해야 한다.
- ✎ 추출되는 데이터 행이 없거나 하나 이상인 경우 예외가 발생한다.
- ✎ 여러 개의 행을 하나씩 추출해야 하는 경우는 명시적 커서(**Explicit Cursor**)를 사용한다.
- ✎ **SQL** 문은 세미콜론(;)으로 끝나야 한다.

□ PL/SQL

● 한 행만을 검색하도록 하지 않으면 에러가 발생하여 **PL/SQL** 블록수행이 종료됨

* 발생하는 에러의 종류

| 조 건 | 예 외 |
|----------------------------|---|
| SELECT 문이 한 행 이상을 추출하는 경우 | TOO_MANY_ROWS 예외(다수 행 검색) (Oracle Server 오류번호 : 01422) |
| SELECT 문이 아무 행도 추출하지 않는 경우 | NO_DATA_FOUND 예외(검색 행이 없는 경우) (Oracle Server 오류번호 : 01403) |

* 에러 해결

- ☑ **Exception Heading** 처리 루틴을 이용하여 처리
- ☑ 명시적(**explicit**) 커서를 선언하여 루프에서 한 행씩 여러 행을 추출하도록 함으로써 해결.

□ PL/SQL

● INSERT 문 사용하기 * SQL의 INSERT 문과 동일.

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE PROCEDURE Insert_Test
2  ( v_emp_no  IN salary.emp_no%TYPE,
3    v_ename   IN salary.emp_name%TYPE,
4    v_sal     IN salary.sal%TYPE,
5    v_deptno  IN salary.dept_no%TYPE)
6  IS
7  BEGIN
8
9      DBMS_OUTPUT.ENABLE;
10
11     INSERT INTO salary(emp_no, emp_name, sal, dept_no)
12     VALUES(v_emp_no, v_ename, v_sal, v_deptno);
13
14     DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_emp_no );
15     DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_ename );
16     DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || v_sal );
17     DBMS_OUTPUT.PUT_LINE( '사원부서 : ' || v_deptno );
18     DBMS_OUTPUT.PUT_LINE( '데이터 입력 성공' );
19
20 END ;
21 /

프로시저가 생성되었습니다.

SQL> |
```

SELECT 문을 이용하여 데이터 값을
먼저 조회한 후 조회된 값을 사용하여
INSERT 문으로 데이터를 삽입하는 예제

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> EXECUTE INSERT_TEST(1004, 'HONG', 3000000, 44);
사원번호 : 1004
사원이름 : HONG
사원급여 : 3000000
사원부서 : 44
데이터 입력 성공

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> SELECT * FROM SALARY;

   EMP_NO EMP_NAME      SAL      DEPT_NO
-----
   1001  PARK        1500000         11
   1002  KIM         2000000         22
   1003  LEE         2400000         33
   1004  HONG         3000000         44

SQL>
```

□ PL/SQL

● UPDATE 문 사용하기

```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> CREATE OR REPLACE PROCEDURE Update_Test
  2  ( v_empno IN    salary.emp_no%TYPE,      -- 급여를 수정한 사원의 사번
  3    v_rate   IN    NUMBER )                -- 급여의 인상/인하율
  4
  5  IS
  6
  7  -- 수정 데이터를 확인하기 위한 변수 선언
  8  v_emp  salary%ROWTYPE ;
  9
 10  BEGIN
 11
 12      DBMS_OUTPUT.ENABLE;
 13
 14      UPDATE salary
 15      SET sal = sal+(sal * (v_rate/100))  -- 급여를 계산
 16      WHERE emp_no = v_empno ;
 17
 18      DBMS_OUTPUT.PUT_LINE( '데이터 수정 성공 ' );
 19
 20  -- 수정된 데이터 확인하기 위해 검색
 21  SELECT emp_no, emp_name, sal
 22  INTO v_emp.emp_no, v_emp.emp_name, v_emp.sal
 23  FROM salary
 24  WHERE emp_no = v_empno ;
 25
 26      DBMS_OUTPUT.PUT_LINE( ' **** 수 정 확 인 **** ');
 27      DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_emp.emp_no );
 28      DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_emp.emp_name );
 29      DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || v_emp.sal );
 30
 31  END ;
 32 /

프로시저가 생성되었습니다.

SQL>
```

특정 사원의 SALARY 테이블의
급여를 조정하는 프로시저

```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> SET SERVEROUTPUT ON ;
SQL> EXECUTE UPDATE_TEST(1002, 50);
데이터 수정 성공
**** 수 정 확 인 ****
사원번호 : 1002
사원이름 : KIM
사원급여 : 3000000

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>
```

□ PL/SQL

● DELETE 문 사용하기

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE PROCEDURE Delete_Test
2   ( p_empno IN salary.emp_no%TYPE )
3
4   IS
5
6   -- 삭제 데이터를 확인하기 레코드 선언
7   TYPE del_record IS RECORD
8   ( v_empno      salary.emp_no%TYPE,
9     v_ename      salary.emp_name%TYPE,
10    v_deptno      salary.dept_no%TYPE);
11
12   v_salary      del_record ;
13
14   BEGIN
15
16       DBMS_OUTPUT.ENABLE;
17
18   -- 삭제된 데이터 확인용 쿼리
19       SELECT emp_no, emp_name, dept_no
20       INTO v_salary.v_empno, v_salary.v_ename, v_salary.v_deptno
21       FROM salary
22       WHERE emp_no = p_empno ;
23
24       DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_salary.v_empno );
25       DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_salary.v_ename );
26       DBMS_OUTPUT.PUT_LINE( '입 사 일 : ' || v_salary.v_deptno);
27
28   -- 삭제 쿼리
29       DELETE
30       FROM salary
31       WHERE emp_no = p_empno ;
32
33       DBMS_OUTPUT.PUT_LINE( '데이터 삭제 성공 ' );
34
35   END;
36   /

```

프로시저가 생성되었습니다.

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> SELECT * FROM SALARY;

```

| EMP_NO | EMP_NAME | SAL | DEPT_NO |
|--------|----------|---------|---------|
| 1001 | PARK | 1500000 | 11 |
| 1002 | KIM | 3000000 | 22 |
| 1003 | LEE | 2400000 | 33 |
| 1004 | HONG | 3000000 | 44 |

삭제 프로시저 실행 전

↓

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> EXECUTE DELETE_TEST(1004);
사원번호 : 1004
사원이름 : HONG
입 사 일 : 44
데이터 삭제 성공
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL> SELECT * FROM SALARY;

```

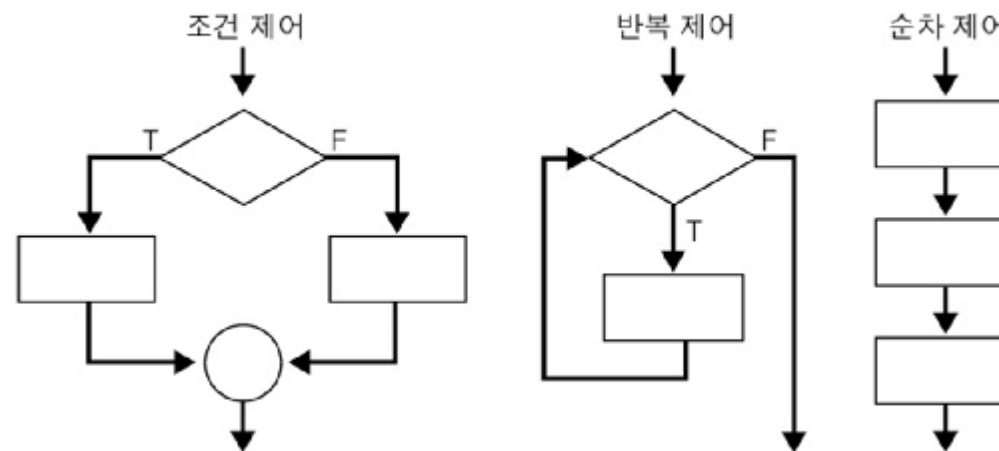
| EMP_NO | EMP_NAME | SAL | DEPT_NO |
|--------|----------|---------|---------|
| 1001 | PARK | 1500000 | 11 |
| 1002 | KIM | 3000000 | 22 |
| 1003 | LEE | 2400000 | 33 |

삭제 프로시저 실행 후

□ PL/SQL

● PL/SQL 제어문

- * 문장들의 논리적 흐름을 변경
- * 종류
 - ☑ 조건 제어(Conditional Control)
 - ☑ 반복 제어(Iterative Control)
 - ☑ 순차 제어(Sequential Control)



PL/SQL 블록의 제어 구조

● 조건 제어 (Conditional Control)

* 조건에 따라 선택적으로 작업을 수행하도록 하는 구문

- ☑ 조건에 따라 참인 경우와 거짓인 경우 각각 다른 문장을 수행하는 구조
- ☑ IF 문과 CASE 문

* IF 문

- ☑ 조건이 TRUE이면 THEN 이하의 문장을 실행하고, 조건이 FALSE나 NULL이면 ELSE 이하의 문장 실행
- ☑ 복수의 ELSIF 절을 사용 가능, ELSE 절은 하나만 사용해야 한다.
- ☑ IF~THEN 문, IF~THEN~ELSE 문, IF~THEN~ELSIF 문

□ PL/SQL

* IF ~ THEN 문

- ☑ **PL/SQL** 블록이 조건이 참(**TRUE**)인 경우에만 조건문을 실행하는 구문
- ☑ 조건이 거짓(**FALSE**)이거나 **NULL**이면 **PL/SQL**은 조건문을 무시
- ☑ 조건이 참인 경우나 거짓인 경우, 어느 경우에도 제어는 **END IF** 다음의 문장에서 시작.

```
IF 조건문 THEN
    조건이 참인 경우 실행할 문장들 ;
END IF;
```

* IF ~ THEN ~ ELSE 문

- ☑ 조건이 **TRUE**이면 **THEN** 이하의 문장을 실행하고, 조건이 **FALSE**나 **NULL**이면 **ELSE** 이하의 문장을 실행

```
IF 조건문 THEN
    조건이 참인 경우 실행할 문장들 ;
ELSE
    조건이 거짓인 경우 실행할 문장들 ;
END IF;
```

□ PL/SQL

- ✎ THEN 절과 ELSE 절 안에 또 다른 IF 문을 중첩하여 사용 가능
 - 중첩된 IF문은 END IF와 반드시 짝을 이루어야 함

☑ IF ~ THEN ~ ELSIF 문

- ✎ 조건이 참과 거짓 두 경우로만 나뉘지 않고 경우의 수가 2개 이상인 경우에 사용하는 제어 구조

```
IF 조건문 THEN
    조건문이 참인 경우 실행할 문장들 ;
ELSIF 조건문_1 THEN
    조건문_1 이 참인 경우 실행할 문장들 ;
ELSE
    위 조건이 모두 거짓인 경우 실행할 문장들 ;
END IF;
```

□ PL/SQL

* CASE 문

- ☑ 조건에 따라 실행할 문장을 선택
- ☑ 실행할 문장은 **CASE** 절에 명시된 선택자(selector)에 의해 이루어짐

```
[<<label_name>>]  
CASE selector  
    WHEN expression1 THEN  
    WHEN expression2 THEN  
    ...  
    WHEN expressionN THEN  
    [ELSE sequence_of_statementsN+ ]  
END CASE [label_name];
```

□ PL/SQL

● CASE문 예제

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';

  CASE
    WHEN grade = 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN grade = 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN grade = 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN grade = 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN grade = 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
  END CASE;
EXCEPTION
  WHEN CASE_NOT_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No such grade');
END;
```

□ PL/SQL

● 조건절에서 연산자 사용하기

* IS NULL 연산자

☑ 널(UNKNOWN) 값 처리

☑ IS NULL 비교의 결과 참(TRUE)이나 거짓(FALSE) 반환

* 논리 연산자를 이용한 **boolean** 연산

☑ 비교 연산자를 사용하여 하나 이상의 조건 포함

☑ **AND, OR, NOT**

☑ **NULL**과 비교연산시 결과는 항상 **UNKNOWN**

| | | | |
|---------------|---------|---------|---------|
| AND (min) | TRUE | FALSE | UNKNOWN |
| TRUE (1) | TRUE | FALSE | UNKNOWN |
| FALSE (0) | FALSE | FALSE | FALSE |
| UNKNOWN (1/2) | UNKNOWN | FALSE | UNKNOWN |
| OR (max) | TRUE | FALSE | UNKNOWN |
| TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | UNKNOWN |
| UNKNOWN | TRUE | UNKNOWN | UNKNOWN |

| | |
|-------|-------|
| NOT | |
| TRUE | FALSE |
| FALSE | TRUE |
| NULL | NULL |

□ PL/SQL

● 반복 제어 (Iterative Control)

* 특징

- ☑ 한 문장 또는 일련의 문장들을 반복 실행할 수 있는 루프(**Loop**)를 구성하기 위한 유형
- ☑ 기본(**BASIC**) 루프, **FOR** 루프, **WHILE** 루프

* 기본(**BASIC**) 루프

- ☑ **LOOP**와 **END LOOP** 사이에 반복되는 문장 부분들로 구성
- ☑ 실행상의 흐름이 **END LOOP**에 도달할 때마다 그와 짝을 이루는 **LOOP** 문으로 제어가 되돌아간다. => 무한 루프
- ☑ 루프에서 빠져나가려면 **EXIT** 문 사용.

```
LOOP
    실행할 문장들;
    EXIT [ WHEN 조건절 ];
END LOOP;
```


□ PL/SQL

● EXIT 문

- * **END LOOP** 문 다음 문으로 제어를 보내기 때문에 루프 종료
- * **IF** 문 내의 처리 작업으로서, 또는 루프 내의 독립적인 문장으로서도 사용 가능
- * 조건에 따라 루프를 종료할 수 있도록 **WHEN** 절 추가 가능
- * **label**을 포함하면 **label**로 표시된 루프를 빠져나옴

```
LOOP
    ...
    IF credit_rating < 3 THEN
        ...
        EXIT; -- 이 절을 만나면 바로 루프를 빠져 나가게 됨
    END IF;
END LOOP;
```

```
LOOP
    FETCH c1 INTO ...
    EXIT WHEN c1%NOTFOUND; -- 조건이 참이면 루프를 빠져나감
    ...
END LOOP;
CLOSE c1;
```

□ PL/SQL

● FOR 루프

* PL/SQL이 수행할 반복 횟수를 정하기 위한 제어문 가짐

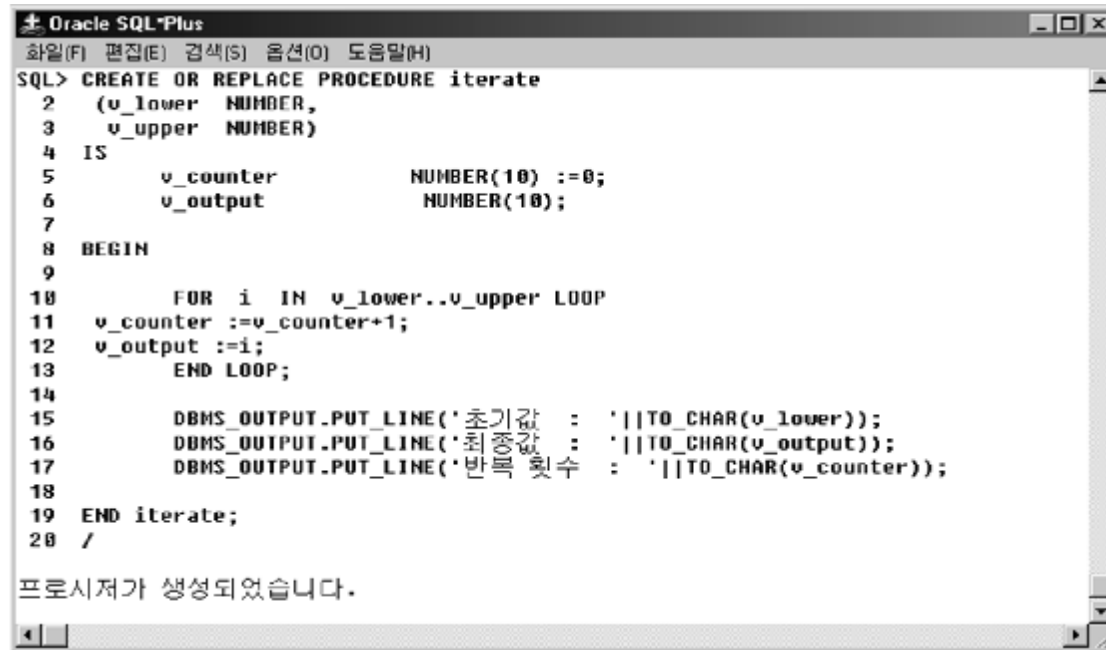
```
FOR 인덱스 IN [ REVERSE ] 하한..상한 LOOP
    문장1;
    문장2;
    ...
END LOOP;
```

- ☑ 인덱스는 상한에 도달할 때까지 루프를 반복할 때마다 자동으로 1씩 증감하는 값으로 정수
- ☑ 인덱스는 정수로 자동 선언되므로 따로 선언할 필요가 없다.
- ☑ **REVERSE**는 상한에서 하한까지 인덱스가 반복 때마다 감소

```
FOR i IN REVERSE 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
END LOOP;
```

- ☑ **IN** 다음에는 커서(cursor)나 **select** 문이 올 수 있다.

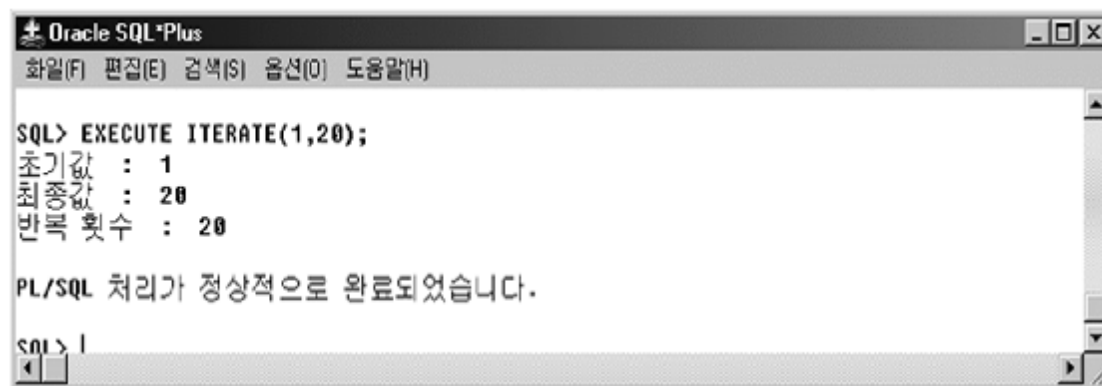
□ PL/SQL



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE PROCEDURE iterate
2   (v_lower NUMBER,
3    v_upper NUMBER)
4   IS
5       v_counter    NUMBER(10) :=0;
6       v_output      NUMBER(10);
7
8   BEGIN
9
10      FOR i IN v_lower..v_upper LOOP
11          v_counter :=v_counter+1;
12          v_output :=i;
13      END LOOP;
14
15      DBMS_OUTPUT.PUT_LINE('초기값   : '||TO_CHAR(v_lower));
16      DBMS_OUTPUT.PUT_LINE('최종값   : '||TO_CHAR(v_output));
17      DBMS_OUTPUT.PUT_LINE('반복 횟수 : '||TO_CHAR(v_counter));
18
19  END iterate;
20  /

프로시저가 생성되었습니다.
```

반복 횟수 계산하는 FOR ~ LOOP 문



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> EXECUTE ITERATE(1,20);
초기값   : 1
최종값   : 20
반복 횟수 : 20

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> |
```

실행 결과

□ PL/SQL

● WHILE 루프

- * 제어 조건이 참(**TRUE**)인 동안만 문장을 반복하게 하는 경우 사용
 - ☑ 반복되는 첫 문장에서 매번 조건을 평가하여 조건이 거짓(**FALSE**)이 되면 루프 종료
 - ☑ 루프의 시작에서 조건이 거짓(**FALSE**)이면 루프를 더 이상 실행하지 않는다.

* WHILE 루프의 구문

```
WHILE 조건 LOOP
    문장1;
    문장2;
    ...
END LOOP;
```

□ PL/SQL

● 순차 제어

* GOTO 문과 NULL 문

☑ NULL 문

- ✎ 제어 구조 내에서 어떤 행동도 하지 않을 것을 명령하는 것으로 조건문의 의미를 읽기 쉽게 하고 분명하게 하기 위한 목적으로 많이 사용

☑ GOTO 문

- ✎ 프로그램 구조를 복잡하고 이해하기 어렵게 하므로 **GOTO** 문보다는 예외 처리를 하는 것이 좋다.

* GOTO 문

- ☑ 프로그램 수행 중에 **GOTO** 문을 만나면 제어가 **GOTO** 문에 명시되어 있는 레이블로 분기

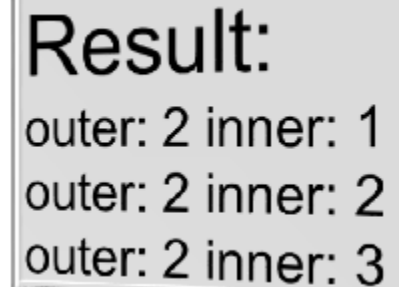
☑ 제한

- ✎ **IF, ELSE** 문, 루프문, 서브 블록등의 내부로 가도록 지정 불가
- ✎ 서브 프로그램 블록 밖으로 건너뛰도록 지정 불가
- ✎ 예외 절에서 예외 절이 사용되었던 **PL/SQL** 블록 안으로 다시 들어올 수 없다.

□ PL/SQL

● Label 사용 예

```
BEGIN
  <<outer_loop>>
  FOR i IN 1..3 LOOP
    <<inner_loop>>
    FOR i IN 1..3 LOOP
      IF outer_loop.i = 2 THEN
        DBMS_OUTPUT.PUT_LINE
          ('outer: ' || TO_CHAR(outer_loop.i) || ' inner: '
           || TO_CHAR(inner_loop.i));
      END IF;
    END LOOP inner_loop;
  END LOOP outer_loop;
END;
```



Result:
outer: 2 inner: 1
outer: 2 inner: 2
outer: 2 inner: 3

□ PL/SQL

● NULL 문

- * 데이터 값의 **NULL**과 달리 제어가 이동한 부분에서 처리해야 할 내용이 아무것도 없다는 것을 분명하게 표시하기 위해 사용
- * 예

```
IF TOTAL > 90 THEN  
    compute_grade( student_no ) ;  
ELSE  
    NULL;  
END IF;
```

□ PL/SQL

● 커서 (Cursor)

* 특징

- ☑ 특정 **SELECT**나 **DML** 문장을 처리하기 위해 필요한 정보를 저장하는 **Private SQL** 영역에 대한 포인터
- ☑ 입력 장치의 현재 위치를 가리키며,
- ☑ 공유 메모리 영역(**SGA**)에 존재하는 **SQL**에 접근할 수 있도록 하는 기법 의미.

처리 결과

| | | | | |
|------|------|-------|---------|------|
| 커서 → | 7369 | SMITH | CLERK | 현재 행 |
| | 7566 | JONES | MANAGER | |
| | 7788 | SCOTT | ANALYST | |
| | 7876 | ADAMS | CLERK | |
| | 7902 | FORD | ANALYST | |

☑ 종류

✎ 명시적 커서(Explicit Cursor)

- 모든 **SQL** 문을 실행할 때 문맥을 가리키는 포인터로서의 역할을 수행

✎ 암시적 커서(Implicit Cursor)

- **SELECT**나 **DML** 문장을 실행할 때마다 자동으로 열리는 커서

● 암시적 커서 (Implicit Cursor)

* 특징

- ☑ 오라클이나 **PL/SQL** 실행 메커니즘에 의해 처리되는 **SQL** 문에 대한 익명의 주소
- ☑ 오라클 데이터베이스에서 실행되는 모든 **SQL** 문은 암시적인 커서
- ☑ **SQL** 문이 실행되는 순간 자동으로 열리고, 닫힘

* 암시적 커서의 속성

- ☑ **SQL%ROWCOUNT** : 해당 **SQL** 문에 영향을 받는 행의 수
- ☑ **SQL%FOUND** : 해당 **SQL** 영향을 받는 행의 수가 1개 이상일 경우 참
- ☑ **SQL%NOTFOUND** : 해당 **SQL** 문에 영향을 받는 행의 수가 없을 경우 참
- ☑ **SQL%ISOPEN** : 항상 **FALSE**, 암시적 커서가 열려 있는지 여부 검색

□ PL/SQL

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE PROCEDURE Implicit_Cursor
2  (p_empno    salary.emp_no%TYPE)
3
4  IS
5
6  v_sal    salary.sal%TYPE;
7  v_update_row  NUMBER;
8
9  BEGIN
10
11  SELECT sal
12  INTO v_sal
13  FROM    salary
14  WHERE emp_no = p_empno ;
15
16  -- 검색된 데이터가 있을경우
17  IF SQL%FOUND THEN
18
19    DBMS_OUTPUT.PUT_LINE('검색한 데이터가 존재합니다 : '||v_sal);
20
21  END IF;
22
23  UPDATE salary
24  SET sal = sal*1.1
25  WHERE emp_no = p_empno;
26
27  SELECT sal
28  INTO v_sal
29  FROM    salary
30  WHERE emp_no = p_empno ;
31
32
33  -- 수정한 데이터의 카운트를 변수에 저장
34  v_update_row := SQL%ROWCOUNT;
35
36
37  DBMS_OUTPUT.PUT_LINE('급여가 인상된 직원 수 : '|| v_update_row);
38  DBMS_OUTPUT.PUT_LINE('인상된 급여 : '|| v_sal);
39
40  END;
41  /

프로시저가 생성되었습니다.
```

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> EXECUTE IMPLICIT_CURSOR(1002);
검색한 데이터가 존재합니다 : 3300000
급여가 인상된 직원 수 : 1
인상된 급여 : 3630000

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> /
```

□ PL/SQL

● SQL%NOTFOUND 속성의 문제점

- * **SELECT * INTO ...로 실행시 검색 결과가 없는 경우 NO_DATA_FOUND exception이 먼저 raise 됨**

☑ IF SQL%NOTFOUND THEN으로의 확인은 불가능

● 함수와 프로시저어 테스트 방법

- * 함수

```
SELECT get_length('star wars', 1977) FROM DUAL;
```

- * 프로시저어

```
VAR out1 INTEGER;
```

```
VAR out2 VARCHAR2(255);
```

```
EXECUTE Get_Std_Info('star wars', 1977, :out1, :out2);
```

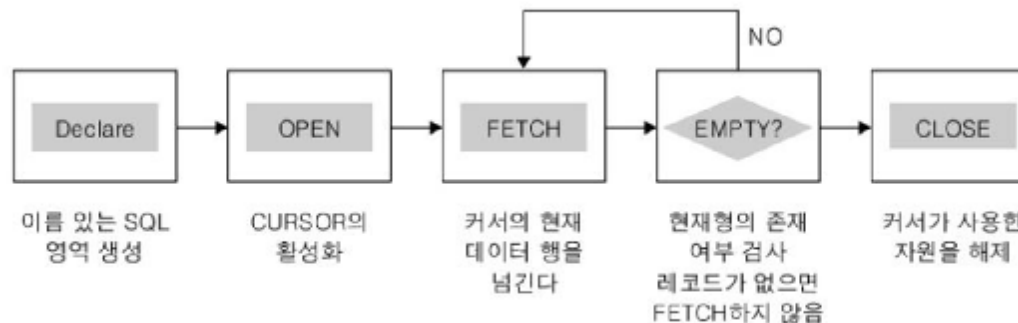
```
PRINT out1;
```

```
PRINT out2;
```

□ PL/SQL

● 명시적 커서 (Explicit Cursor)

※ PL/SQL 블록에서 프로그래머가 커서를 제어하도록 함



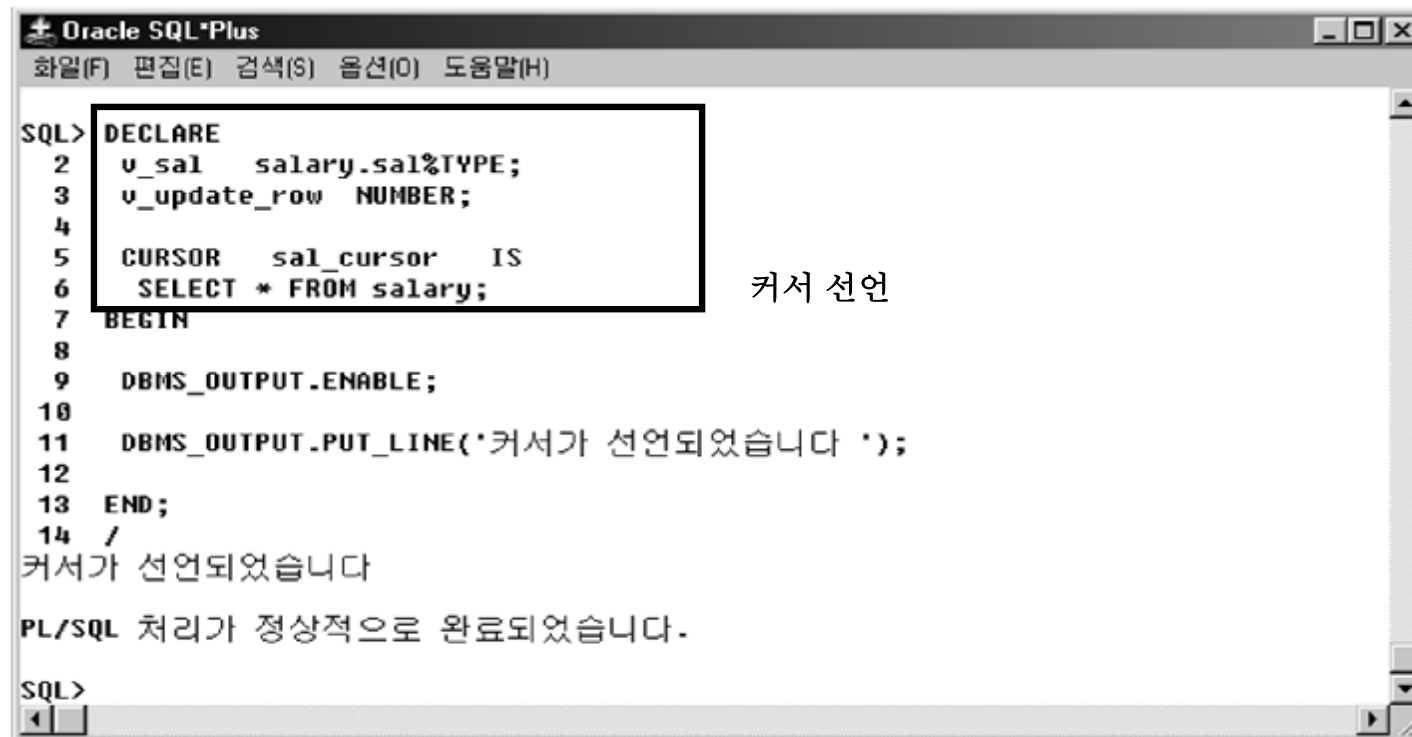
명시적 커서의 제어 단계

※ 커서 선언하기

- ☑ 커서는 사용하기 전에 먼저 선언되어야 함
- ☑ 선언된 커서는 한 개의 이름이 할당되고 **SELECT** 문과 연결됨.
- ☑ 커서 선언 내에는 **INTO** 절을 쓰지 않는다.
- ☑ 커서 선언문의 구조

```
DECLARE  
    CURSOR 커서이름 IS  
        select 문 ;
```

□ PL/SQL



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> DECLARE
2   v_sal salary.sal%TYPE;
3   v_update_row NUMBER;
4
5   CURSOR sal_cursor IS
6     SELECT * FROM salary;
7 BEGIN
8
9   DBMS_OUTPUT.ENABLE;
10
11   DBMS_OUTPUT.PUT_LINE('커서가 선언되었습니다 ');
12
13 END;
14 /
커서가 선언되었습니다
PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>
```

커서 선언

□ PL/SQL

● 커서 열기(OPEN)

- * OPEN 문 사용
- * 커서 안의 검색이 실행되며 아무런 데이터 행을 추출하지 못하면 에러 발생

```
OPEN 커서이름 ;
```

● 커서 패치(FETCH)

- * 현재 데이터 행을 **OUTPUT** 변수에 반환
- * 커서의 **SELECT** 문의 컬럼 수와 **OUTPUT** 변수의 수가 동일해야 함.
 - ☑ 커서 컬럼의 변수 타입과 **OUTPUT** 변수의 데이터 타입도 동일해야 함.
- * 한 행씩 데이터 패치.
- * 커서 패치 구문

```
FETCH 커서이름 INTO 변수_1, 변수_2 ;
```

□ PL/SQL

● 커서 닫기(CLOSE)

- * 사용을 마친 커서는 다른 변수의 이름으로 다시 열 수 있도록 반드시 닫아 주어야 한다.
- * 커서를 닫은 상태에서 **FETCH**를 할 수 없다.

```
CLOSE 커서이름 ;
```

□ PL/SQL

☑ 커서를 이용한 데이터 처리하기

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE PROCEDURE Cursor_Test
2      (v_dept_no salary.dept_no%TYPE)
3
4  IS
5
6      CURSOR salary_avg IS
7          SELECT dept_no, COUNT(emp_no), ROUND(AVG(sal))
8          FROM salary
9          WHERE dept_no = v_dept_no
10         GROUP BY dept_no ;
11
12  -- 커서를 패치하기 위한 변수 선언하기
13  v_deptno salary.dept_no%TYPE;
14  emp_cnt  NUMBER;
15  sal_avg  NUMBER;
16
17  BEGIN
18
19  -- 커서 오픈
20      OPEN salary_avg;
21
22  -- 커서 패치
23      FETCH salary_avg INTO v_deptno, emp_cnt, sal_avg ;
24
25      DBMS_OUTPUT.PUT_LINE('부서명 : ' || v_deptno);
26      DBMS_OUTPUT.PUT_LINE('직원수 : ' || emp_cnt);
27      DBMS_OUTPUT.PUT_LINE('평균급여 : ' || sal_avg);
28
29  -- 커서의 CLOSE
30      CLOSE salary_avg;
31
32  EXCEPTION
33
34      WHEN OTHERS THEN
35
36          DBMS_OUTPUT.PUT_LINE(SQLERRM || '에러 발생 ');
37
38  END;
39  /

```

프로시저가 생성되었습니다.

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> EXECUTE CURSOR_TEST(22);
부서명 : 22
직원수 : 4
평균급여 : 2997500
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

실행 결과

□ PL/SQL

* FOR 문에서 커서 사용하기

- ☑ **FOR LOOP** 문을 사용하면 커서의 **OPEN, FETCH, CLOSE**가 자동 발생하므로 따로 기술할 필요가 없으며, 레코드 이름도 자동 선언되므로 따로 선언할 필요가 없다

```
FOR 레코드이름 IN 커서이름 LOOP
    문장1;
    문장2;
    ...
END LOOP;
```

□ PL/SQL

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE PROCEDURE ForCursor_Test
  2  (v_deptno    salary.dept_no%TYPE)
  3  IS
  4
  5      CURSOR    sal_sum IS
  6          SELECT dept_no, COUNT(emp_no) emp_cnt, SUM(sal) salary
  7          FROM    salary
  8          WHERE   dept_no = v_deptno
  9          GROUP BY dept_no;
 10
 11 BEGIN
 12
 13  -- Cursor를 FOR문에서 실행
 14  FOR    sal_list IN    sal_sum    LOOP
 15
 16      DBMS_OUTPUT.PUT_LINE('부서명 : ' || sal_list.dept_no);
 17      DBMS_OUTPUT.PUT_LINE('사원수 : ' || sal_list.emp_cnt);
 18      DBMS_OUTPUT.PUT_LINE('급여합계 : ' || sal_list.salary);
 19
 20  END LOOP;
 21
 22  EXCEPTION
 23
 24  WHEN OTHERS THEN
 25      DBMS_OUTPUT.PUT_LINE(SQLERRM || '에러 !!!!! ');
 26
 27  END;
 28  /
```

프로시저가 생성되었습니다.

SQL> |

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> EXECUTE FORCOURSE_TEST(33);
부서명 : 33
사원수 : 2
급여합계 : 5300000

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> |
```

● **CURSOR** 사용 예 : **%ROWTYPE** 사용

```
DECLARE
    m Movie%ROWTYPE;
    CURSOR c1 IS SELECT * FROM Movie;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO m;
        EXIT WHEN c1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(m.title||','||m.year);
    END LOOP;
    CLOSE c1;
END;
```

● **CURSOR** 사용 예 : %**TYPE** 사용

DECLARE

t Movie.title%TYPE;

y Movie.year%TYPE;

CURSOR c1 IS SELECT title, year FROM Movie;

BEGIN

OPEN c1;

LOOP

FETCH c1 INTO t, y;

EXIT WHEN c1%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(t||''||y);

END LOOP;

CLOSE c1;

END;

□ PL/SQL

● CURSOR 사용 예 : BULK COLLECT 사용

```
DECLARE
    TYPE TitleTab IS TABLE OF Movie.title%TYPE;
    TYPE YearTab IS TABLE OF Movie.year%TYPE;
    ts TitleTab;
    ys YearTab;
    CURSOR c1 IS SELECT title, year FROM Movie WHERE length > 100;
BEGIN
    OPEN c1;
        FETCH c1 BULK COLLECT INTO ts, ys;
    CLOSE c1;
    FOR i IN ts.FIRST .. ts.LAST LOOP
        IF ys(i) > 1970 THEN
            DBMS_OUTPUT.PUT_LINE( ts(i) || ' is after 1970');
        END IF;
    END LOOP;
    FOR i IN ys.FIRST .. ys.LAST LOOP
        IF ts(i) LIKE '%and%' THEN
            DBMS_OUTPUT.PUT_LINE( ts(i) || '('||ys(i)||') includes the word "and"');
        END IF;
    END LOOP;
END;
```

□ PL/SQL

● CURSOR 사용 예 : 파라미터 사용

```
DECLARE
  CURSOR c1 (yy Movie.year%TYPE) IS
    SELECT * FROM Movie WHERE year > yy;
BEGIN
  FOR m IN c1(1980) LOOP
    -- process data record
    DBMS_OUTPUT.PUT_LINE('영화제목 : ' || m.title || ', 연도 : ' ||
      m.year || ', 영화사 : ' || m.studioName );
  END LOOP;
END
```

□ *PL/SQL*

● **CURSOR** 사용 예 : 파라미터 사용

```
DECLARE
  m Movie%ROWTYPE;
  CURSOR c1 (yy Movie.year%TYPE, word VARCHAR2) IS
    SELECT * FROM Movie WHERE year > yy and title > word;
BEGIN
  -- OPEN c1(1990, 'USA');
  -- OPEN c1(1950, 'XYZ');
  OPEN c1(1990, 'korea');
  LOOP
    FETCH c1 INTO m;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('제목 : ' || m.title ||', 연도: ' || m.year ||', 영화사 : ' || m.studioName );
  END LOOP;
END
```

□ PL/SQL

● CURSOR 사용 예 : WHERE CURRENT OF 사용

DECLARE

m Movie%ROWTYPE;

CURSOR c1 (yy Movie.year%TYPE, word VARCHAR2) IS

SELECT * FROM Movie WHERE year > yy and title > word

FOR UPDATE;

BEGIN

-- OPEN c1(1990, 'USA'); 또는 OPEN c1(1950, 'XYZ');

OPEN c1(1990, 'korea');

LOOP

FETCH c1 INTO m;

EXIT WHEN c1%NOTFOUND;

DBMS_OUTPUT.PUT_LINE('영화제목 : ' || m.title || ', 연도 : ' || m.year || ', 영화사 : ' || m.studioName);

IF m.year > 1995 THEN

UPDATE Movie SET length = 500 WHERE CURRENT OF c1;

ELSIF m.length = 100 THEN

DELETE FROM Movie WHERE CURRENT OF c1;

END IF;

END LOOP;

END

□ *PL/SQL*

● Dynamic SQL

* Static vs. Dynamic

- ☑ static SQL : compile때 SQL을 처리, DDL 문 처리 불가, 효율적, 비유동적(*inflexible*)
- ☑ dynamic SQL : run-time에 SQL 문 처리, 비효율적, 유동적

● DBMS_SQL 패키지 사용

```
CREATE OR REPLACE PROCEDURE utilities.drop_table (  
    schema_name IN OUT VARCHAR2, table_name IN OUT VARCHAR2) IS  
    cursor_id INTEGER;  
    return_value INTEGER;  
    command_string VARCHAR2(250);  
BEGIN  
    command_string := 'DROP TABLE ' || schema_name || '.' || table_name;  
    cursor_id := dbms_sql.open_cursor;  
    dbms_sql.parse(cursor_id, command_string, dbms_sql.v7);  
    return_value := dbms_sql.execute(cursor_id);  
    dbms_sql.close_cursor(cursor_id);  
END drop_table;
```

□ *PL/SQL*

● Dynamic SQL

* Static vs. Dynamic

- ☑ static SQL : compile때 SQL을 처리, DDL 문 처리 불가, 효율적, 비유동적(*inflexible*)
- ☑ dynamic SQL : run-time에 SQL 문 처리, 비효율적, 유동적

● DBMS_SQL 패키지 사용

```
CREATE OR REPLACE PROCEDURE utilities.drop_table (  
    schema_name IN OUT VARCHAR2, table_name IN OUT VARCHAR2) IS  
    cursor_id INTEGER;  
    return_value INTEGER;  
    command_string VARCHAR2(250);  
BEGIN  
    command_string := 'DROP TABLE ' || schema_name || '.' || table_name;  
    cursor_id := dbms_sql.open_cursor;  
    dbms_sql.parse(cursor_id, command_string, dbms_sql.v7);  
    return_value := dbms_sql.execute(cursor_id);  
    dbms_sql.close_cursor(cursor_id);  
END drop_table;
```

● EXECUTE IMMEDIATE 문장

```
DECLARE
  sql_str  VARCHAR2(500);
  p_name   PEOPLE.NAME%TYPE := '경성대';
  p_birth  PEOPLE.BIRTHDATE%TYPE := '1984-01-01';
BEGIN
  sql_str := 'INSERT INTO PEOPLE VALUES (:1, :2, PHONE_TAB(), ADDR_TAB())';
  EXECUTE IMMEDIATE sql_str USING p_name, p_birth;
  sql_str := 'INSERT INTO TABLE(SELECT PHONE_LIST FROM PEOPLE WHERE NAME =
    :1) VALUES (phone_ty("office", 1, "01012341234"))';
  EXECUTE IMMEDIATE sql_str USING p_name;
  sql_str := 'INSERT INTO TABLE(SELECT ADDRESSES FROM PEOPLE WHERE NAME =
    :1) VALUES (addr_ty("서울", "동작", "노량진"))';
  EXECUTE IMMEDIATE sql_str USING p_name;
  COMMIT;
END;
```

□ PL/SQL

● EXECUTE IMMEDIATE 문장

DECLARE

```
sql_stmt VARCHAR2(100);  
my_deptno NUMBER(2) := 50;  
my_dname VARCHAR2(15) := 'PERSONNEL';  
my_loc VARCHAR2(15) := 'DALLAS';  
emp_rec emp%ROWTYPE;
```

BEGIN

```
sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';  
EXECUTE IMMEDIATE sql_stmt USING my_deptno, my_dname, my_loc;  
sql_stmt := 'SELECT * FROM emp WHERE empno = :id';  
EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING 7788;  
EXECUTE IMMEDIATE 'DELETE FROM dept WHERE deptno = :n'  
USING my_deptno;
```

...

```
EXECUTE IMMEDIATE 'CREATE TABLE bonus (id NUMBER, amt NUMBER)';  
sql_stmt := 'ALTER SESSION SET SQL_TRACE TRUE';  
EXECUTE IMMEDIATE sql_stmt;
```

END;

□ PL/SQL

● Dynamic SQL을 이용한 PL/SQL 예

```
DECLARE
tbl VARCHAR2(100);
sql_ex VARCHAR2(200);
CURSOR c1 IS SELECT TABLE_NAME FROM USER_TABLES;
BEGIN
OPEN c1;
LOOP
FETCH c1 INTO tbl;
EXIT WHEN c1%NOTFOUND;
sql_ex := 'DROP TABLE ' || tbl || ' CASCADE CONSTRAINTS';
BEGIN
    EXECUTE IMMEDIATE sql_ex;
    DBMS_OUTPUT.PUT_LINE(tbl||' was dropped...');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Cannot Drop '||tbl);
END;
END LOOP;
CLOSE c1;
END;
```

● OPEN FOR 문장

DECLARE

TYPE EmpCurTyp IS REF CURSOR; -- define weak REF CURSOR type

emp_cv EmpCurTyp; -- declare cursor variable

my_ename VARCHAR2(15);

my_sal NUMBER := 1000;

sql_string VARCHAR2(50);

BEGIN

sql_string := 'SELECT ename, sal FROM emp WHERE sal > ***:salary***'

-- open cursor variable

OPEN emp_cv FOR sql_string USING my_sal;

LOOP

 FETCH emp_cv INTO my_ename, my_sal; -- fetch next row

 EXIT WHEN emp_cv%NOTFOUND;

 ...

END LOOP;

CLOSE emp_cv; -- close cursor variable

...

□ PL/SQL

● OPEN FOR 문장

```
DECLARE
    TYPE MovieCT IS REF CURSOR;
    movie_cv MovieCT;
    m_title movie.title%TYPE;
    my_year movie.year%TYPE := 1980;
    s_name movie.studioname%TYPE;
    sql_string VARCHAR(255);
BEGIN
    sql_string := 'SELECT title, studioname FROM movie WHERE year > :year';
    OPEN movie_cv FOR sql_string USING my_year;
    LOOP
        FETCH movie_cv INTO m_title, s_name;
        EXIT WHEN movie_cv%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('제목 : '||m_title||'..영화사 : '||s_name);
    END LOOP;
    CLOSE movie_cv;
END;
```

□ PL/SQL

● 애트리뷰트로 **Nested Table** 사용

※ 하나 이상의 전화번호를 애트리뷰트에 저장

```
CREATE TYPE phone_ty AS OBJECT (  
  name VARCHAR2(20),  
  seq  INTEGER,  
  no   CHAR(11)  
);
```

```
CREATE OR REPLACE TYPE phone_tab AS TABLE OF phone_ty;
```

```
CREATE TABLE person (  
  name VARCHAR2(20) PRIMARY KEY,  
  birthdate DATE,  
  phone_list phone_tab)  
  NESTED TABLE phone_list STORE AS phone_table;
```


□ PL/SQL

● Person 테이블에 튜플 삽입

```
insert into person values (  
    '홍석희',  
    '1990-01-01',  
    phone_tab( phone_ty('mobile', 1, '01012346789') ) );
```

```
insert into table (select phone_list from person where name = '홍석희')  
values (phone_ty('office', 2, '0516635678'));  
insert into table (select phone_list from person where name = '홍석희')  
values (phone_ty('home', 3, '0546635140'));
```

```
insert into person values ( '강남길', '1953-10-06', phone_tab() );  
insert into table (select phone_list from person where name = '강남길')  
values (phone_ty('office', 1, '0516062145'));  
insert into table (select phone_list from person where name = '강남길')  
values (phone_ty('home', 2, '0546074870'));
```

□ PL/SQL

● Person 테이블에 대한 질의

```
SELECT p.name, p.birthdate, l.no  
FROM person p,  
      TABLE(SELECT phone_list FROM person WHERE name = '강남길') l  
WHERE p.name = '강남길';
```

```
SELECT p.name, p.birthdate, l.no  
FROM person p, TABLE(p.phone_list) l  
WHERE p.name = '강남길';
```

| NAME | BIRTHDATE | NO |
|------|------------|------------|
| 강남길 | 1953-10-06 | 0516062145 |
| 강남길 | 1953-10-06 | 0546074870 |

```
SELECT p.name, COUNT(l.no)  
FROM person p, TABLE(p.phone_list) l  
GROUP BY p.name;
```

| NAME | COUNT(L.NO) |
|------|-------------|
| 강남길 | 2 |
| 홍석희 | 3 |

□ PL/SQL

● 영화 정보를 위한 **Nested Table** 생성

```
create type star_ty as object (  
    name      varchar2(30),  
    salary    number  
);
```

```
create type studio_ty as object (  
    name      varchar2(30),  
    investment number  
);
```

```
create or replace type star_tab as table of star_ty;  
create or replace type studio_tab as table of studio_ty;  
create table MovieInfo (  
    title  varchar2(255),  
    year   smallint,  
    length smallint,  
    stars  star_tab,  
    studios studio_tab)  
nested table stars store as s_table  
nested table studios store as st_table;
```

□ PL/SQL

● 예외(Exceptions)

※ 오라클 PL/SQL의 오류

※ PL/SQL을 컴파일할 때 문법적인 오류로 발생하는 컴파일 타임 오류와 프로그램을 실행할 때 발생하는 실행 타임 오류로 구분

☑ 컴파일 타임 오류 : 오라클 PL/SQL 컴파일러에 의해 발생

☑ 실행 타임 오류 : 오라클 PL/SQL 엔진에 의해 오류 여부 검색

| 예외 종류 | 설 명 | 처 리 |
|----------------------|----------------------------|-----------------------------------|
| 미리 정의된 오라클 서버 에러 | 오라클 PL/SQL 코드에서 자주 발생하는 오류 | 선언할 필요 없이 예외절에서 자동으로 트랩(Trap) |
| 미리 정의되지 않은 오라클 서버 에러 | 오라클 서버 오류를 제외하고 미리 정의된 오류 | 선언부에서 선언해야 하고 오류가 발생하면 자동으로 트랩 |
| 사용자 정의 에러 | 사용자가 설정한 오류 처리 | 선언부에서 선언하고 실행부에서 RAISE 문을 사용하여 발생 |

오라클 PL/SQL 오류의 종류

□ PL/SQL

● 예외절 구조

* 예외절

- ☑ 예외를 명확하게 함으로써 프로그램 구조를 이해하기 쉽도록 함
- ☑ 명시된 오류가 발생하는 경우 프로그램을 중지하기보다는 예외절에 의해 처리되도록 함.

* 예외절 구조

```
EXCEPTION
  WHEN 예외_1 [ OR 예외_2 ... ] THEN
    실행문_1; ...
  WHEN 예외_3 [ OR 예외_4 ... ] THEN
    실행문_2; ...
  WHEN OTHERS THEN
    실행문_2; ...
```

- ✍️ 마지막에 하나의 **WHEN OTHERS** 문 사용 가능
- ✍️ 여러 개의 예외 처리부(Exception Handler)를 허용하여 예외가 발생하면 여러 개의 예외 처리부 중에 하나의 예외 처리부에 트랩(Trap)

□ PL/SQL

* 미리 정의된 예외

| 예외 | 오라클 에러 (SQLCODE) | 설명 |
|---------------------|---------------------|--|
| ACCESS_INTO_NULL | ORA-06530(-6530) | 초기화되지 않은 객체에 값을 할당하는 경우 |
| CASE_NOT_FOUND | ORA-06592(-6592) | CASE 문에 ELSE 절이 없는 경우 |
| CURSOR_ALREADY_OPEN | ORA-06511(-6511) | 이미 열려있는 커서를 다시 열려고 하는 경우 |
| DUP_VAL_ON_INDEX | ORA-00001(-1) | UNIQUE 제약을 가지는 컬럼에 중복되는 데이터를 삽입하려고 하는 경우 |
| INVALID_CURSOR | ORA-01001(-1001) | 잘못된 커서 연산을 수행하려고 하는 경우 |
| INVALID_NUMBER | ORA-01722(-1722) | 잘못된 숫자를 표현한 경우 |
| LOGIN_DENIED | ORA-01017(-1017) | 사용자 아이디와 암호를 가지고 오라클에 로그인하는 경우 |
| NO_DATA_FOUND | ORA-01403(+100) | SELECT 문이 반환할 데이터 행이 없는 경우 |
| NOT_LOGGED_ON | ORA-01012(-1012) | 오라클에 연결되지 않은 데이터베이스를 호출하는 경우 |
| PROGRAM_ERROR | ORA-06501(-6501) | 내부 PL/SQL 오류 |

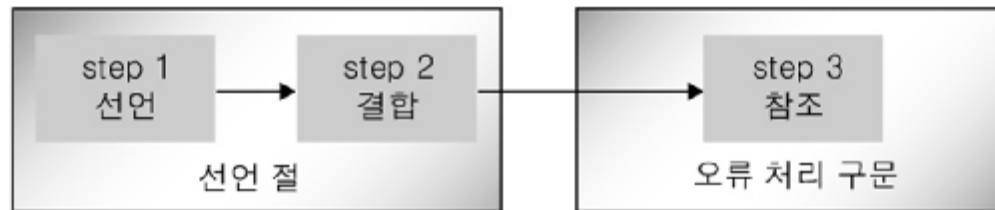
□ PL/SQL

| 예외 | 오라클 에러 (SQLCODE) | 설명 |
|-------------------------|---------------------|---|
| SELF_IS_NULL | ORA-30625(-30625) | NULL 인스턴스에 대해 MEMBER 메소드를 호출한 경우 |
| STORAGE_ERROR | ORA-06500(-6500) | 메모리 부족으로 일어나는 PL/SQL 내부 오류 |
| SUBSCRIPT_BEYOND_COUNT | ORA-06533(-6533) | 엘리먼트의 수보다 큰 인덱스를 가지고 중첩된 테이블이나 변수를 호출한 경우 |
| SUBSCRIPT_OUTSIDE_LIMIT | ORA-06532(-6532) | 범위 이외의 수를 가지고 중첩된 테이블이나 변수를 호출한 경우 |
| TIMEOUT_ON_RESOURCE | ORA-00051(-51) | 자원을 기다리는 동안 TIME-OUT이 발생한 경우 |
| TOO_MANY_ROWS | ORA-01422(-1422) | SELECT 문이 하나 이상의 행을 반환하는 경우 |
| VALUE_ERROR | ORA-06502(-6502) | 숫자의 계산, 변환, 버림 등에서 발생한 오류 |
| ZERO_DIVIDE | ORA-01476(-1476) | 0으로 나누려 하는 경우 |

□ PL/SQL

* 미리 정의되지 않은 오류

- ☑ 오라클의 오류 중에서 미리 정의되지 않은 오류를 정의하여 사용하려면 다음 단계로 처리



미리 정의되지 않은 오류 처리

- ❶ 예외 이름을 선언절에서 선언
- ❷ 선언절에서 PRAGMA EXCEPTION_INIT 문장으로 예외의 이름과 오라클 서버 오류 번호를 결합
- ❸ 실행절에서 예외가 발생한 경우 해당 예외를 참조

□ PL/SQL

```

Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> DECLARE
2  insert_not_row EXCEPTION;
3  PRAGMA EXCEPTION_INIT( insert_not_row, -00001);
4
5  BEGIN
6  DBMS_OUTPUT.ENABLE;
7  INSERT INTO SALARY VALUES(1001, 'ABC', 123456, 22);
8  EXCEPTION
9  WHEN insert_not_row THEN
10 DBMS_OUTPUT.PUT_LINE ( '해당 값 중복 ' );
11 END;
12 /
해당 값 중복

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>

```

미리 정의되지 않은 오류 처리

- * 예외절에서 **WHEN OTHERS** 문으로 트랩되는 에러들의 실제 에러 코드와 그 설명을 실제로 확인하고 싶으면 **SQLCODE**와 **SQLERRM**을 사용

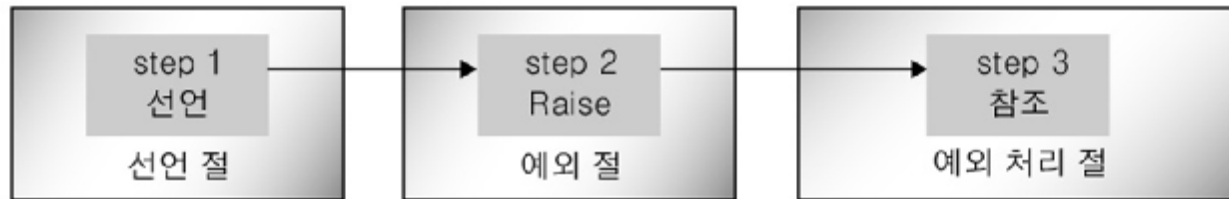
| SQLCODE 값 | 설명 |
|-----------|------------------------|
| 0 | 오류 없이 성공적으로 종료 |
| 1 | 사용자 정의 예외 번호 |
| +100 | NO_DATA_FOUND 예외 번호 |
| 음수 | 위에 것을 제외한 오라클 서버 에러 번호 |

SQLCODE 값

□ PL/SQL

* 사용자 정의 예외

- ☑ 사용자는 오라클 저장 함수 **RAISE_APPLICATION_ERROR**를 사용하여 에러 코드 **-20000**부터 **-20999**의 범위 내에서 사용자 정의 예외 생성 가능
- ☑ 생성 단계



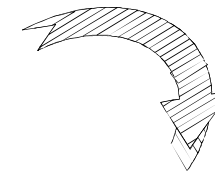
- ❶ 선언절에서 예외 이름 선언
- ❷ RAISE 문을 사용하여 실행절에서 예외를 직접적으로 발생시킴
- ❸ 예외절에서 예외가 발생할 경우 예외를 참조

□ PL/SQL

```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> CREATE OR REPLACE PROCEDURE User_Defined_Exception
 2      (v_deptno IN salary.dept_no%type )
 3  IS
 4
 5  -- 예외의 이름을 선언
 6  user_define_error EXCEPTION; -- 1단계(예외 이름 선언)
 7
 8      cnt      NUMBER;
 9
10 BEGIN
11
12      DBMS_OUTPUT.ENABLE;
13
14      SELECT COUNT(emp_no)
15      INTO cnt
16      FROM salary
17      WHERE dept_no = v_deptno;
18  IF cnt < 5 THEN
19      RAISE user_define_error;
20      -- 2단계( RAISE문을 사용하여 직접 예외를 발생)
21  END IF;
22
23  EXCEPTION
24      WHEN user_define_error THEN -- 3단계( 예외가 발생할 경우 해당
25
26      RAISE_APPLICATION_ERROR(-20001, '부서에 사원이 몇명 안되네요..');
27
28 END;
29 /
```

프로시저가 생성되었습니다.



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL>
SQL> EXECUTE USER_DEFINED_EXCEPTION(22);
BEGIN USER_DEFINED_EXCEPTION(22); END;

*
1행에 오류:
ORA-20001: 부서에 사원이 몇명 안되네요..
ORA-06512: "SYS.USER_DEFINED_EXCEPTION", 줄 26에서
ORA-06512: 줄 1에서

SQL> |
```

□ PL/SQL

- 예외 처리 후 계속 실행하기
 - * 일반적인 예외 발생시 해당 프로시주어나 블록은 예외 처리 후 실행 종료 함
 - * 내장된 블록에 예외 처리시 해당 블록만 종료 됨
- 예제

```
BEGIN
  DBMS_OUTPUT.ENABLE;
  FOR i IN 1..20 LOOP
    BEGIN
      SELECT * INTO exec
      FROM MovieExec
      WHERE i = certNo;
      IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('제작자 이름 : '||exec.Name);
      END IF;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        NULL;
    END;
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Runtime Error');
END;
```

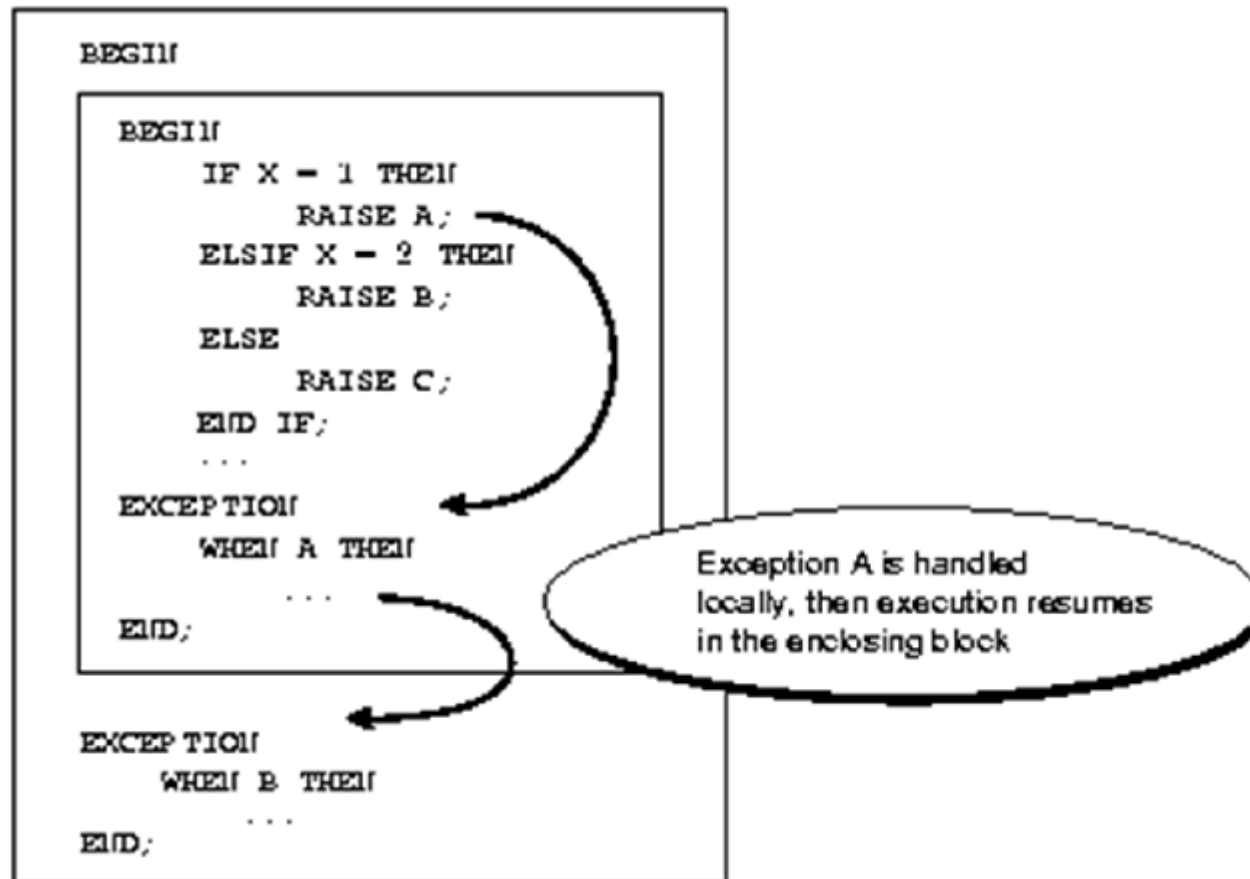
□ PL/SQL

● 사용자 정의 예외처리 사용 예

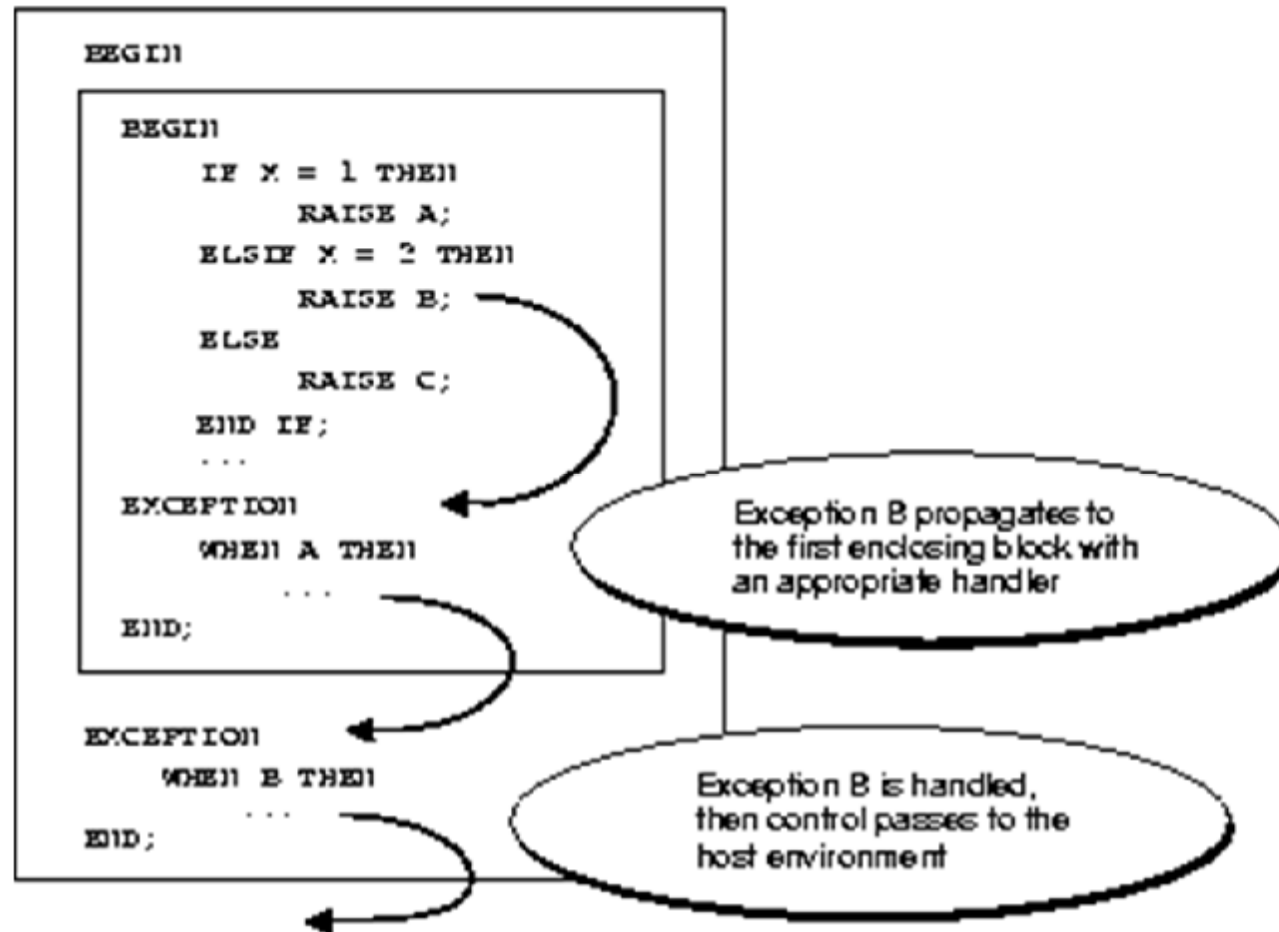
```
DECLARE
    no_movie_star  EXCEPTION;
    cnt  integer;
BEGIN
    FOR ex IN (select * from movieexec) LOOP
        BEGIN
            SELECT count(*) INTO cnt
            FROM starsin
            WHERE ex.name = starname;
            IF cnt = 0 THEN
                RAISE  no_movie_star;
            END IF;
            DBMS_OUTPUT.PUT_LINE('제작자 '||ex.name||'은 '||cnt||'편의 영화에 출연 함');
        EXCEPTION
            WHEN no_movie_star THEN
                -- RAISE_APPLICATION_ERROR(-20001, '배우가 아님');
                DBMS_OUTPUT.PUT_LINE('제작자 '||ex.name||'은 영화에 출연한 적이 없음...');
        END;
    END LOOP;
END;
```

□ PL/SQL

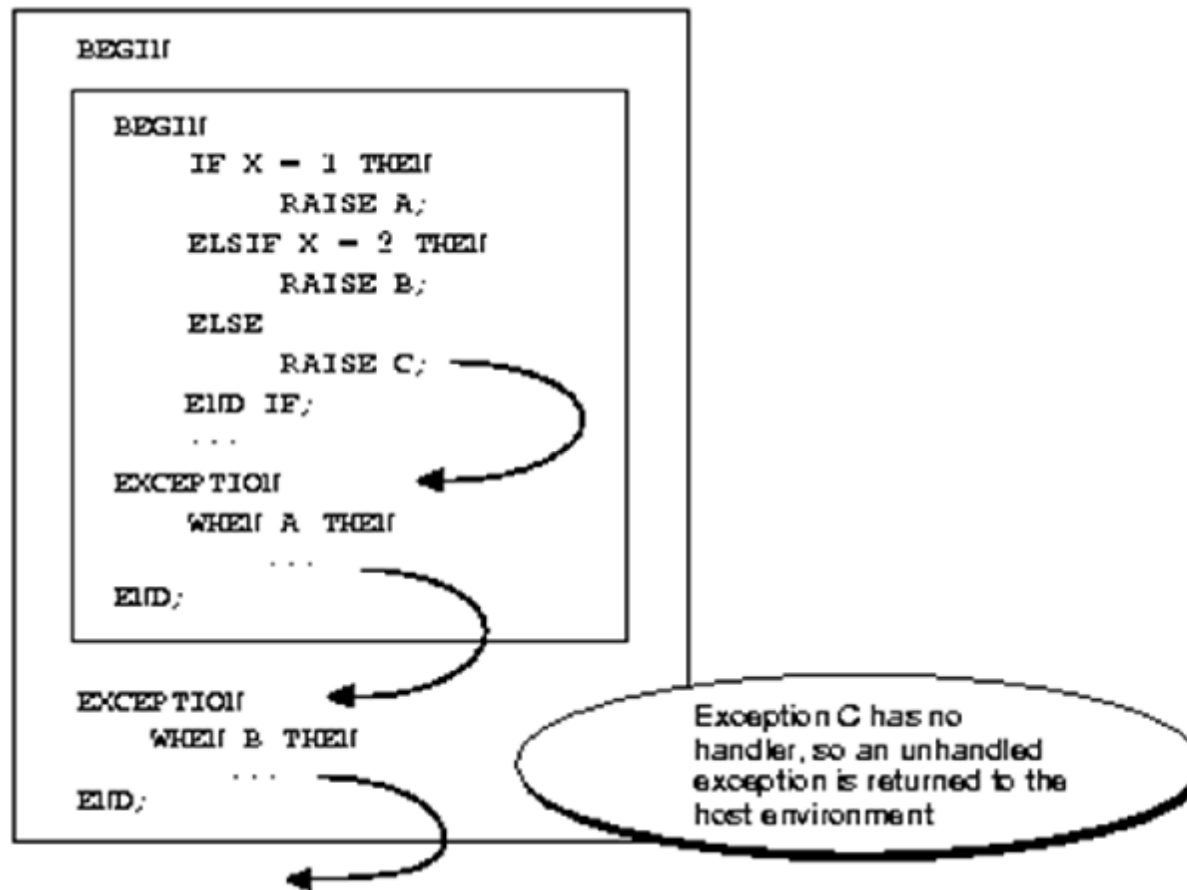
● Exception Propagation Rule 1



● Exception Propagation Rule 2



● Exception Propagation Rule 3



□ *PL/SQL*

● Reraising

```
DECLARE
    out_of_balance EXCEPTION;
BEGIN
    ...
    BEGIN ----- sub-block begins
        ...
        IF ... THEN
            RAISE out_of_balance; -- raise the exception
        END IF;
    EXCEPTION
        WHEN out_of_balance THEN -- handle the error
            RAISE; -- reraise the current exception
    END; ----- sub-block ends
EXCEPTION
    WHEN out_of_balance THEN    -- handle the error differently
    ...
END;
```

● Exception Propagation : OUT-OF-SCOPE

BEGIN

...

DECLARE ----- sub-block begins

past_due **EXCEPTION**; -- local exception

BEGIN

...

IF ... THEN

RAISE *past_due*;

END IF;

END; ----- sub-block ends

EXCEPTION

...

WHEN OTHERS THEN

ROLLBACK;

END;

} **sub-block**

□ *PL/SQL*

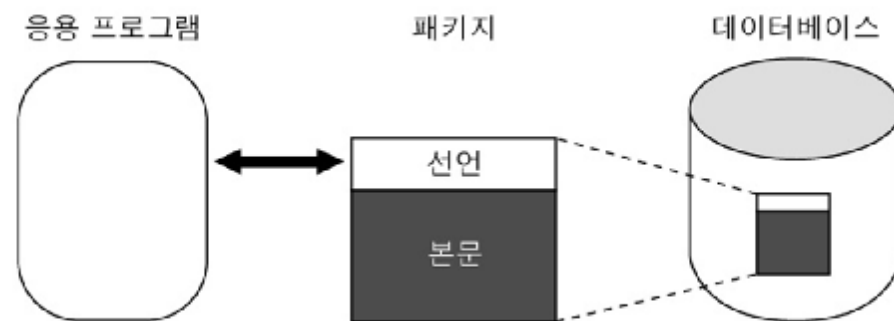
● **SQLCODE와 SQLERRM : SQL 에러코드와 메시지 출력**

```
DECLARE
    err_num NUMBER;
    err_msg VARCHAR2(100);
    k NUMBER;
BEGIN
    k := 10 / 0;
EXCEPTION
    WHEN OTHERS THEN
        err_num := SQLCODE;
        err_msg := SUBSTR(SQLERRM, 1, 100);
        DBMS_OUTPUT.PUT_LINE(err_num || ' : ' || err_msg);
END;
```

□ PL/SQL

● 패키지(package)

- * 오라클 데이터베이스에 저장되어 있는 서로 관련있는 **PL/SQL** 타입, 항목, 프로시저, 함수의 집합
- * 선언부와 본문 두 부분으로 구성



패키지 인터페이스

□ PL/SQL

● 패키지 선언부

* 특징

- ☑ 패키지에 포함될 **PL/SQL** 프로시저나, 함수, 커서, 변수, 예외절 등을 선언
- ☑ 패키지 선언부에서 선언한 모든 요소는 패키지 전체에 적용
 - ✍ 선언부에서 선언한 변수는 **PUBLIC** 변수로 사용
- ☑ 패키지 선언부의 선언은 **CREATE PACKAGE** 또는 **CREATE OR REPLACE PACKAGE** 문 사용

```
CREATE [OR REPLACE] PACKAGE 패키지이름
    [ AUTHID { CURRENT_USER | DEFINER } ]
    { IS | AS }
    [ PRAGMA SERIALLY_REUSABLE; ]
    [ collection_type_definition ... ]
    [ record_type_definition ... ]
    [ subtype_definition ... ]
    [ collection_declaration ... ]
    [ constant_declaration ... ]
    [ exception_declaration ... ]
    [ object_declaration ... ]
    [ record_declaration ... ]
    ...
END [ 패키지이름 ];
```

□ PL/SQL

● 패키지 선언부

```
CREATE PACKAGE trans_data AS -- bodiless package
    TYPE TimeRec IS RECORD (
        minutes SMALLINT,
        hours SMALLINT);
    TYPE TransRec IS RECORD (
        category VARCHAR2(10),
        account INT,
        amount REAL,
        time_of TimeRec);
    minimum_balance CONSTANT REAL := 10.00;
    number_processed INT;
    insufficient_funds EXCEPTION;
END trans_data;
```

□ PL/SQL

● 패키지 본문

- * 패키지에서 선언된 요소의 실행을 정의하는 부분
- * 선언된 요소의 실제 코드를 작성하는 부분
- * **CREATE PACKAGE BODY** 또는 **CREATE OR REPLACE PACKAGE BODY** 문을 사용하여 정의

```
[ CREATE [ OR REPLACE ] PACKAGE BODY 패키지이름 { IS | AS }  
  [ PRAGMA SERIALLY_REUSABLE; ]  
  [ collection_type_definition ... ]  
  [ record_type_definition ... ]  
  [ subtype_definition ... ]  
  [ collection_declaration ... ]  
  [ constant_declaration ... ]  
  [ exception_declaration ... ]  
  [ object_declaration ... ]  
  [ record_declaration ... ]  
  [ variable_declaration ... ]  
  [ cursor_body ... ]  
  [ function_spec ... ]  
  [ procedure_spec ... ]  
  [ call_spec ... ]  
  [ BEGIN  
      sequence_of_statements ]  
END [ 패키지이름 ]; ]
```

□ *PL/SQL*

● Package Body 생성 예

```
CREATE OR REPLACE PACKAGE BODY emp_mgmt AS
    tot_emps NUMBER;
    tot_depts NUMBER;

    FUNCTION create_dept(department_id NUMBER, location_id NUMBER)
        RETURN NUMBER IS
        new_deptno NUMBER;
    BEGIN
        SELECT ...
    END;

    PROCEDURE remove_emp (employee_id NUMBER) IS
    BEGIN
        DELETE ...
    END;

    ...
    PROCEDURE increase_comm(employee_id NUMBER, comm_incr NUMBER) IS
        curr_comm NUMBER;
    BEGIN
        SELECT ..
    END;

END emp_mgmt;
```


□ PL/SQL

● 패키지 사용

- * package_name.type_name
- * package_name.item_name
- * package_name.subprogram_name
- * package_name.call_spec_name

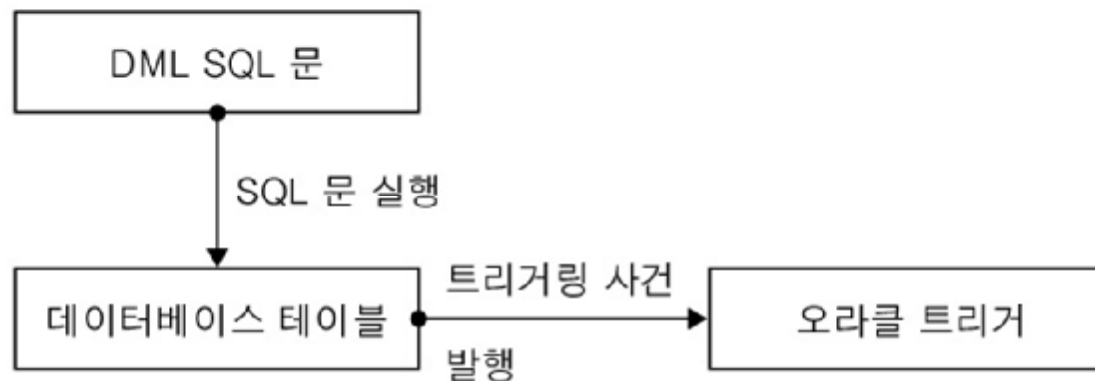
emp_actions.hire_employee(e_id,e_lname,e_fname, ...);

□ PL/SQL

● 오라클 트리거(Trigger)

* 특징

- ☑ 어떤 사건이 발생했을 때 내부적으로 실행되도록 데이터베이스에 저장된 프로시저
- ☑ 선언절, 실행절, 예외절을 가지는 **PL/SQL** 블록 구조를 가지고 데이터베이스에 저장되어야 한다
- ☑ 트리거링 사건(Triggering Event) : DML 문장(DELETE, INSERT, UPDATE), DDL 문장(CREATE, ALTER, DROP), DB 연산(SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN)



● 오라클 트리거 사용 범위

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

□ PL/SQL

● 트리거 생성하기

* 특징

- ☑ **CREATE TRIGGER** 문을 사용하여 생성
- ☑ 트리거를 생성하기 위해서는 **CREATE TRIGGER** 권한을 가지고 있어야 함.

* 문법의 구조

```
CREATE [ OR REPLACE ] TRIGGER 트리거이름 BEFORE | AFTER  
  Triggering_event ON 테이블이름  
  [ FOR EACH ROW ]  
  [ WHEN (조건식) ]  
  PL/SQL 블록;
```

□ PL/SQL

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> CREATE OR REPLACE TRIGGER salary_changes
2 BEFORE DELETE OR INSERT OR UPDATE ON emp_table
3 FOR EACH ROW
4 WHEN (new.Emp_no > 0)
5 DECLARE
6   sal_diff number;
7 BEGIN
8   sal_diff := :new.salary - :old.salary;
9   DBMS_OUTPUT.PUT_LINE('이전 급여 : ' || :old.salary);
10  DBMS_OUTPUT.PUT_LINE('새로운 급여 : ' || :new.salary);
11  DBMS_OUTPUT.PUT_LINE('급여 차액 ' || sal_diff);
12  END;
13 /

트리거가 생성되었습니다.
```

트리거 실행 결과

```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)

SQL> UPDATE EMP_TABLE
2 SET SALARY=SALARY+SALARY*0.15
3 WHERE DEPT_NO=22;

이전 급여 : 3400000
새로운 급여 : 3910000
급여 차액 510000
이전 급여 : 2000000
새로운 급여 : 2300000
급여 차액 300000
이전 급여 : 2000000
새로운 급여 : 2300000
급여 차액 300000

3 행이 갱신되었습니다.
```

□ PL/SQL

● 문장 트리거와 행 트리거

* FOR EACH ROW 옵션 생략시 문장 트리거(Statement-Level Trigger)

- ☑ 트리거링 사건에 대해 딱 한번만 실행
- ☑ WHEN 절 사용 불가
- ☑ 컬럼의 각 데이터 행을 제어할 수 없다. (:old, :new 등 사용 불가)
 - ✍ 컬럼의 데이터 값에 관계없이 컬럼에 변화가 일어남을 감지하여 실행되는 트리거

* FOR EACH ROW 옵션 사용시 행 트리거(Row-Level Trigger)

- ☑ 컬럼의 각 데이터 행이 변경될 때마다 실행
- ☑ 실제 그 데이터 행의 값을 제어할 수 있는 트리거
- ☑ 실제 값을 수정, 변경 또는 저장하기 위해 사용
- ☑ 각 행은 :OLD와 :NEW로 지칭 됨
- ☑ SQL 문에 따른 사용 방법
 - ✍ INSERT 문의 경우 - 입력할 데이터의 값은 “:NEW.컬럼이름”
 - ✍ UPDATE 문의 경우 - 변경 전의 데이터는 “:OLD.컬럼이름”, 새로운 데이터 값은 “:NEW.컬럼이름”
 - ✍ DELETE 문의 경우 - 삭제되는 컬럼 값은 “:OLD.컬럼이름”

□ PL/SQL

● INSTEAD OF에 대한 제약 :

* 뷰에 대해서만 사용 가능

☑ 집합 연산자, **DISTINCT**, 집계(**aggregation**) 함수, **GROUP BY**, **ORDER BY**, **SELECT** 절의 부질의 등을 포함하는 뷰는 제외

* **WHEN** 절 사용 불가

● AFTER/BEFORE에 대한 제약 :

| 시점 | 연산 | 정의 불가 대상 | :NEW 갱신 | :OLD 갱신 |
|--------|--------|----------|---------|---------|
| AFTER | INSERT | 뷰(view) | × | × |
| | DELETE | | | |
| | UPDATE | | | |
| BEFORE | INSERT | | ○ | × |
| | DELETE | | | |
| | UPDATE | | | |

● INSTEAD OF 사용 예

```
CREATE OR REPLACE TRIGGER mv_prod_insert
INSTEAD OF INSERT ON movie_prod
FOR EACH ROW
DECLARE
    exec_id movieexec.certno%TYPE;
BEGIN
    SELECT MAX(certno) INTO exec_id
    FROM movieexec;
    exec_id := exec_id + 1;
    INSERT INTO movieexec(name, certno, networth) VALUES
        (:new.name, exec_id, 100000);
    INSERT INTO movie(title, year, producerno) VALUES
        (:new.title, :new.year, exec_id);
END;
```

```
CREATE VIEW Movie_Prod AS
SELECT title, year, name
FROM Movie, MovieExec
WHERE producerno = certno;
```

```
insert into movie_prod values ('Avengers', 2012, 'Kevin Feige');
```


□ PL/SQL

● Trigger에서 튜플 갱신

- * 사건 대상 테이블의 튜플을 트리거 내에서 갱신시 **DML** 문장 사용 불가

```
CREATE OR REPLACE TRIGGER increasing_length  
BEFORE UPDATE OF length ON movie  
FOR EACH ROW  
BEGIN
```

```
  IF :new.length < :old.length THEN
```

```
    :new.length := :old.length;
```

```
  END IF;
```

```
END;
```

```
UPDATE Movie  
SET  length = length - 30  
WHERE year >= 2000;
```

```
UPDATE Movie  
SET  length = :old.length  
WHERE title = :old.title AND year = :old.year;
```

Mutating Table Error 발생

□ PL/SQL

● 트리거 CASCADING

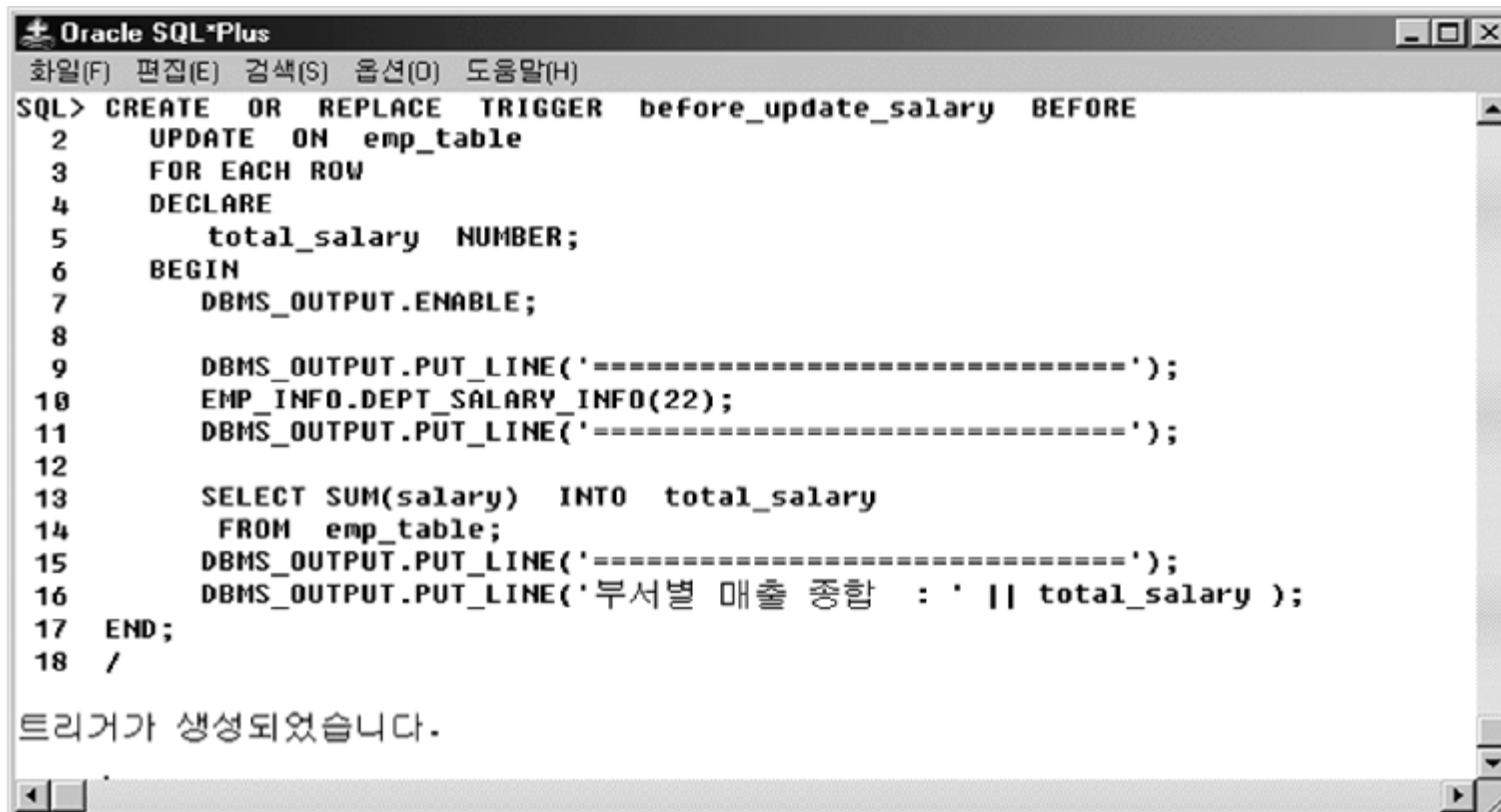
* 정의

- ☑ 한 트리거가 다른 트리거를 실행시키는 것

* 프로시저 또는 함수 사용하기

- ☑ 사용자 정의 함수와 오라클 저장 함수와 데이터베이스에 저장된 모든 프로시저와 함수를 호출 가능
- ☑ 예

✎ 오라클 저장 함수인 **SUM**과 **EMP_INFO** 패키지의 **DEPT_SALARY_INFO** 프로시저를 사용하는 예제

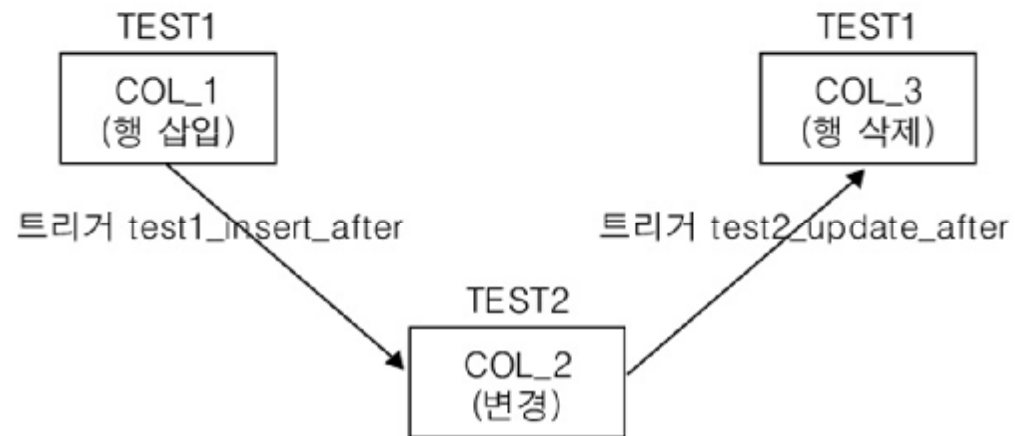


```
Oracle SQL*Plus
화일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> CREATE OR REPLACE TRIGGER before_update_salary BEFORE
2  UPDATE ON emp_table
3  FOR EACH ROW
4  DECLARE
5      total_salary  NUMBER;
6  BEGIN
7      DBMS_OUTPUT.ENABLE;
8
9      DBMS_OUTPUT.PUT_LINE('=====');
10     EMP_INFO.DEPT_SALARY_INFO(22);
11     DBMS_OUTPUT.PUT_LINE('=====');
12
13     SELECT SUM(salary) INTO total_salary
14     FROM emp_table;
15     DBMS_OUTPUT.PUT_LINE('=====');
16     DBMS_OUTPUT.PUT_LINE('부서별 매출 종합 : ' || total_salary );
17 END;
18 /

트리거가 생성되었습니다.
```

□ PL/SQL

☑ 다른 트리거 사용하기 - 트리거 **CASCADING**



TEST1의 COL_1 컬럼에 값이 삽입된 후

TEST2 테이블의 COL_2 컬럼의 데이터를 수정하는 TEST_INSERT_AFTER 트리거와

TEST2 테이블의 컬럼 데이터가 UPDATE되면

TEST3 테이블의 COL_3 데이터를 삭제하는 TEST2_UPDATE_AFTER 트리거 발생

□ PL/SQL

● 트리거 삭제와 활성화/비활성화

* 트리거 삭제

- ☑ **DROP TRIGGER** 문 사용

```
DROP TRIGGER 트리거이름 ;
```

* 트리거 활성화/비활성화

- ☑ **ALTER TRIGGER** 문 사용

```
ALTER TRIGGER 트리거이름 DISABLE ;  
ALTER TRIGGER 트리거이름 ENABLE ;
```

□ PL/SQL

● TRIGGER 예제 1

```
CREATE OR REPLACE TRIGGER order_info_insert  
INSTEAD OF INSERT ON order_info  
DECLARE  
    duplicate_info EXCEPTION;  
    PRAGMA EXCEPTION_INIT (duplicate_info, -00001);  
BEGIN  
    INSERT INTO customers(customer_id, cust_last_name, cust_first_name)  
        VALUES (:new.customer_id, :new.cust_last_name, :new.cust_first_name);  
    INSERT INTO orders (order_id, order_date, customer_id)  
        VALUES (:new.order_id, :new.order_date, :new.customer_id);  
EXCEPTION  
    WHEN duplicate_info THEN  
        RAISE_APPLICATION_ERROR (-20107, 'Dup customer or order ID');  
END order_info_insert;
```

● **TRIGGER** 예 제 2

```
CREATE TRIGGER total_salary
AFTER DELETE OR INSERT OR UPDATE OF department_id, salary ON employees
FOR EACH ROW
BEGIN
    IF DELETING OR (UPDATING AND :old.department_id != :new.department_id) THEN
        UPDATE departments SET total_salary = total_salary - :old.salary
        WHERE department_id = :old.department_id;
    END IF;
    IF INSERTING OR (UPDATING AND :old.department_id != :new.department_id) THEN
        UPDATE departments SET total_salary = total_salary + :new.salary
        WHERE department_id = :new.department_id;
    END IF;
    IF (UPDATING('salary') AND :old.department_id = :new.department_id AND
        :old.salary != :new.salary ) THEN
        UPDATE departments SET total_salary = total_salary - :old.salary + :new.salary
        WHERE department_id = :new.department_id;
    END IF;
END;
```

● TRIGGER 예제 3

```
CREATE OR REPLACE TRIGGER length_networth
BEFORE UPDATE OF LENGTH ON movie
FOR EACH ROW
```

```
DECLARE
```

```
  avrg NUMBER;
```

```
BEGIN
```

```
  SELECT AVG(year)
```

```
  INTO avrg
```

```
  FROM movie
```

```
  WHERE :new.producerno = producerno;
```

PRAGMA AUTONOMOUS_TRANSACTION;

Movie에 대한 UPDATE시 트리거 오류

```
IF avrg < :new.length THEN
```

```
  UPDATE movieexec
```

```
  SET networth = networth + (networth * 0.3)
```

```
  WHERE :new.producerno = certno;
```

```
END IF;
```

```
END;
```

COMMIT;

□ PL/SQL

● 외부

```
CREATE LIBRARY external.odbc as 'c:\windows\system\odbc.dll';

CREATE OR REPLACE FUNCTION external.sql_exec_direct (
-- EXECUTE ANY SQL STATEMENT USING ODBC
    sql_handle BINARY_INTEGER;
    sql_statement VARCHAR2(2000),
    sql_length INTEGER )
RETURN VARCHAR2 AS EXTERNAL
    LIBRARY external.odbc
    NAME SQLExecDirect
    LANGUAGE C;

-- CALLING PROCEDURE
DECLARE
    return_code VARCHAR2(2000);
    stmt VARCHAR2(2000) := 'DELETE FROM access.customers';
BEGIN
    return_code := external.sql_exec_direct(1, stmt, LENGTH(stmt));
    ...
```