6-6. PL/SQL sub program

1. PROCEDURE(프로시저)
2. FUNCTION(내장함수)
3. ORACEL PACKAGE(패키지)
4. TRIGGER(트리거)

1. PROCEDURE (프로시저)

익 명 블 록	서브 프로그램
이름이 지정되지 않은 PL/SQL 블록	이름이 지정된 PL/SQL 블록
매번 사용시마다 컴파일됩니다	최초 실행될 때 한번만 컴파일 됩니다
데이터베이스에 저장되지 않습니다	데이터베이스에 저장됩니다
다른 응용프로그램에서 호출 불가합니다	다른 응용프로그램에서 호출 할 수 있습니다
값을 반환하지 않습니다.	함수일 경우 값을 반환 합니다
파라미터를 사용할 수 없습니다	파라미터를 사용할 수 있습니다

1. PROCEDURE (프로시저)

PROCEDURE (프로시저)

- 프로시저는 지정된 특정 처리를 실행하는 서브 프로그램의 한 유형.
- 단독(standalone)으로 실행되거나 다른 프로시저나 다른 툴(Oracle Developer...) 또는 다른 환경(Pro*C...)등에서 호출되어 실행됨

- 생성 문법

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [( parameter1 [mode1] datatype1,
    parameter2 [mode2] datatype2,
   … )]
IS | AS
PL/SQL Block ;
```

1. PROCEDURE (프로시저)

- Parameter Mode 비교

IN 모드	OUT 모드	IN OUT 모드
기본모드입니다	명시적으로 지정해야 합니다	명시적으로 지정해야 합니다
값이 서브 프로그램에 전달됨	값이 호출환경에 반환됨	값이 서브 프로그램에도 전달되고 호출환경에도 반환됨
형식 파라미터가 상수로 동작	초기화 되지 않은 변수	초기화 된 변수
실제 파라미터가 리터럴, 표현식, 상수 또는 초기화된 변수가 될 수 있습니다.	변수만 사용 가능	변수만 사용 가능
기본값을 할당 할 수 있음	기본값 할당 불가	기본값 할당 불가

1. PROCEDURE (프로시저)

실습 1. 부서번호가 20번 인 사람들의 job 을 'CLERK' 으로 변경하는 프로시저

```
SQL> CREATE OR REPLACE PROCEDURE update_20
  2  IS
  3  BEGIN
  4     UPDATE emp
  5     SET  job='CLERK'
  6     WHERE deptno=20;
  7  END;
  8  /
Procedure created.

SQL> execute update_20;

PL/SQL procedure successfully completed.
```

이 예제는 부서번호가 20 번인 사람들의 JOB 을 CLERK 로 변경하는 프로시저 입니다. 이 프로시저는 사용자로부터 값을 입력 받는 것이 없기 때문에 매개변수를 별도로 사용하지 않고 프로시저를 생성했습니다.
그리고 생성된 프로시저를 실행하려면 EXECUTE 명령어로 프로시저 이름을 적고 실행하면 됩니다.
프로시저를 실행하려면 해당 프로시저의 소유자 이거나 EXECUTE 권한이 있어야 합니다.

1. PROCEDURE (프로시저)

실습 2. 사번을 입력 받아 급여를 인상하는 프로시저

(이번 실습은 사용자로부터 값을 입력 받아야 하므로 입력 매개변수를 사용합니다)

```
SCOTT>SELECT empno , ename, sal FROM emp WHERE empno=7902;

EMPNO      ENAME      SAL
---------- ---------- ----------
    7902       FORD      3000   ← 변경 전 sal 이 3000 입니다.
```

다음 장에 계속....

1. PROCEDURE (프로시저)

```
SCOTT>CREATE OR REPLACE PROCEDURE up_sal
  2  ( vempno   emp.empno%TYPE ) – 입력값을 저장할 변수 vempno 선언
  3  IS
  4  BEGIN
  5   UPDATE emp SET sal=5000
  6   WHERE empno=vempno ;
  7  END;
  8  /

Procedure created.

SCOTT>EXEC   up_sal(7902); -- 프로시저를 실행할 때 사원번호를 함께 입력

PL/SQL procedure successfully completed.
```

위 실습 2번에서 프로시저 생성 구문의 2번 라인을 보면 사용자로부터 값을 입력 받기 위해 Vempno 라는 변수를 선언함을 볼 수 있습니다.
원래는 vempno IN emp.empno%TYPE 로 써야 하지만 IN 모드가 기본이라서 IN 키워드를 생략 한 것입니다.

1. PROCEDURE (프로시저)

실습 3. 사 번을 입력 받아 그 사원의 이름과 급여를 출력하는 프로시저

```
SCOTT>CREATE OR REPLACE PROCEDURE ename_sal
  2  ( vempno   emp.empno%TYPE)
  3  IS
  4      vename   emp.ename%TYPE;
  5      vsal     emp.sal%TYPE;
  6
  7  BEGIN
  8    SELECT ename, sal
  9    INTO   vename , vsal
 10    FROM   emp
 11    WHERE empno=vempno;
 12
 13    DBMS_OUTPUT.PUT_LINE('사원명은 '||vename||' 입니다');
 14    DBMS_OUTPUT.PUT_LINE('급여는 '||vsal||'입니다');
 15  END;
 16  /
```

- 실행 결과 -

```
SCOTT>EXEC ename_sal(7902);

사원명은 FORD 입니다
급여는 5000입니다

PL/SQL procedure successfully completed.
```

1. PROCEDURE (프로시저)

실습 4. OUT 모드 파라미터 사용 예

```
SCOTT>CREATE OR REPLACE PROCEDURE info_prof
  2  ( v_profno     IN     professor.profno%TYPE,
  3    v_name      OUT   professor.name%TYPE, -- 이름값을 저장할 변수
  4    v_pay       OUT   professor.pay%TYPE) – 급여를 저장할 변수
  5  IS
  6  BEGIN
  7    SELECT name, pay INTO v_name , v_pay
  8    FROM   professor
  9    WHERE  profno = v_profno ;
 10  END info_prof ;
 11  /

Procedure created.
```

다음 장에 계속....

1. PROCEDURE (프로시저)

```
SCOTT>DECLARE
  2    v_name   professor.name%TYPE ;
  3    v_pay    professor.pay%TYPE ;
  4  BEGIN
  5    info_prof(1001,v_name , v_pay) ; -- 프로시저를 호출하면서 IN 변수값을 지정.
  6    DBMS_OUTPUT.PUT_LINE(v_name||' 교수의 급여는 '||v_pay||' 입니다.');
  7  END ;
  8  /

조인형 교수의 급여는 550 입니다.

PL/SQL procedure successfully completed.
```

OUT 모드의 변수를
호출해서 사용하는 예

다음 장에 계속....

1. PROCEDURE (프로시저)

```
SCOTT>DECLARE
  2    v_name   professor.name%TYPE ;
  3    v_pay    professor.pay%TYPE ;
  4  BEGIN
  5    info_prof(1001,v_name,v_pay) ;
  6    DBMS_OUTPUT.PUT_LINE('이   름: '||v_name) ;
  7    DBMS_OUTPUT.PUT_LINE('급   여: '||v_pay) ;
  8  END ;
9  /

이   름: 조인형
급   여: 550

PL/SQL procedure successfully completed.
```

OUT 모드의 변수를 호출해서 사용하는 예

다음 장에 계속....

1. PROCEDURE (프로시저)

```
SCOTT>VARIABLE name   VARCHAR2(10)
SCOTT>VARIABLE pay    NUMBER
SCOTT>EXEC info_prof(1001,:name,:pay);

PL/SQL procedure successfully completed.

SCOTT>PRINT name pay

NAME
------------
조인형

       PAY
----------
       550
```

OUT 모드의 변수를 호출해서 사용하는 예

1. PROCEDURE (프로시저)

- PL/SQL 에서 파라미터 이름 지정하는 방식 - 1

```
SCOTT>DECLARE
  2    v_name   professor.name%TYPE ;
  3    v_pay    professor.pay%TYPE ;
  4  BEGIN
  5    info_prof(v_profno =>1001,
  6              v_name => v_name ,
  7              v_pay => v_pay);
  8    DBMS_OUTPUT.PUT_LINE('이름:'||v_name);
  9    DBMS_OUTPUT.PUT_LINE('급여:'||v_pay);
 10  END ;
 11  /

이름:조인형
급여:550

PL/SQL procedure successfully completed.
```

실제 파라미터를 임의의 순서로 나열하며 연관 연산자(=>)를 사용하여 이름 지정 형식 파라미터를 실제 파라미터와 연관시킵니다.

1. PROCEDURE (프로시저)

- PL/SQL 에서 파라미터 이름 지정하는 방식 - 2

```
SCOTT> DECLARE
  2    v_name   professor.name%TYPE ;
  3    v_pay    professor.pay%TYPE ;
  4  BEGIN                              (앞장에서 이어집니다)
  5    info_prof(1001,    -- 첫 번째 위치 변수의 값을 직접 지정
  6              v_name => v_name , -- 이름을 직접 지정함
  7              v_pay => v_pay);
  8    DBMS_OUTPUT.PUT_LINE('이름:'||v_name);
  9    DBMS_OUTPUT.PUT_LINE('급여:'||v_pay);
 10  END ;
 11  /

이름:조인형
급여:550
```

혼합 지정 방식

1. PROCEDURE (프로시저)

실습 5. 생성된 프로시저의 내용을 확인합니다.

USER_SOURCE 딕셔너리를 활용합니다.

```
SCOTT>SELECT text
  2  FROM user_source
  3  WHERE name ='INFO_PROF' ;

TEXT
--------------------------------------------------------------------------------
PROCEDURE info_prof
( v_profno   IN     professor.profno%TYPE,
  v_name   OUT   professor.name%TYPE,
  v_pay     OUT   professor.pay%TYPE)
IS
BEGIN
  SELECT name, pay INTO v_name , v_pay
  FROM   professor
  WHERE   profno = v_profno ;
END info_prof ;

10 rows selected.
```

1. PROCEDURE (프로시저)

프로시저 연습문제 1.

Emp 테이블에 신입 사원의 아래 정보를 입력 받아 등록하는 프로시저를 작성하세요. 신입사원의 정보는 아래와 같습니다.(나머지 컬럼의 값은 null 입니다)
(프로시저 이름과 변수 명은 마음대로 하세요)

* Empno : 4000
* Name : Smith
* Job : Clerk
* Manger no: 7900
* Salary : 3500

1. PROCEDURE (프로시저)

프로시저 연습문제 2.

Emp 테이블을 사용하여 부서번호를 입력 받아서 보너스(comm) 를 아래의 조건으로 계산하는 프로시저를 작성하세요
(프로시저 이름과 변수 명은 적당하게 정하세요)

* 조건
Deptno = 10 이면 급여의 20%
Deptno = 20 이면 급여의 30%
Deptno = 30 이면 급여의 10%
나머지는 0%

1. PROCEDURE (프로시저)

프로시저 연습문제 3.
사원 번호를 입력 받아 삭제 하는 프로시저를 작성하세요.
이 실습을 위해서 emp 테이블을 emp2 로 복사하신 후 emp2 테이블을 사용하여 작업하세요
(프로시저 이름과 변수명은 마음대로 하세요)

프로시저 연습문제 4.
emp , dept 테이블을 사용하여 사원의 사원번호를 입력 받아서 아래처럼 그 사원의 사원번호와 ,이름, 부서명 ,급여, 상여금 을 출력하는 프로시저를 작성하세요.
(프로시저 이름과 변수명은 마음대로 하세요)

사 번 :
이 름 :
부서명 :
급 여 :
상여금 :

2. FUNCTION (내장 함수)

내장 함수와 프로시저는 문법이나 특징이 거의 비슷하지만 차이점은 프로시저는 정해진 작업을 수행한 후 결과를 반환 할 수도 있고(OUT, IN OUT 모드 사용시) 반환 하지 않고 그냥 종료할 수도 있지만 함수는 정해진 작업을 수행 한 후 결과를 돌려준다(RETURN)는 부분임

CREATE [OR REPLACE] FUNCTION function_*name*

[(*parameter1 [mode1] datatype1,*

parameter2 [mode2] datatype2,

…)]

RETURN datatype

IS | AS

PL/SQL Block ;

2. FUNCTION (내장 함수)

함수 예제 1. 부서번호를 입력 받아 최고 급여액을 출력하는 함수

```
SQL> CREATE OR REPLACE FUNCTION s_max_sal
  2     (s_deptno  emp.deptno%TYPE)
  3  return number
  4  IS
  5    max_sal  emp.sal%TYPE ;
  6  BEGIN
  7    SELECT max(sal) INTO max_sal
  8    FROM emp
  9    WHERE deptno=s_deptno;
 10    RETURN max_sal ; -- 이 부분의 데이터 형이 위 3번 줄의 형과 같아야 함
 11  END;
 12  /

Function created.
```

함수 생성 하기

2. FUNCTION (내장 함수)

```
SQL> SELECT  s_max_sal(10) FROM dual;

S_MAX_SAL(10)
-------------------
        5000

SQL> SELECT s_max_sal(20) FROM dual;

S_MAX_SAL(20)
--------------------
        3000
```

함수 사용 하기

2. FUNCTION (내장 함수)

함수 예제 2: 부서번호를 입력 받은 후 해당 부서의 인원수를 구해주는 함수

```
SQL> CREATE OR REPLACE FUNCTION count_mem
  2  ( count   NUMBER )
  3  RETURN   NUMBER
  4  IS
  5     max_count   NUMBER;
  6  BEGIN
  7     SELECT count(*) INTO max_count
  8     FROM emp
  9     WHERE deptno = count;
 10     RETURN max_count;  -- 이부분의 데이터 형이 위 3번줄 데이터 형과 같아야 함
 11   END ;
 12   /

Function created.
```

함수 생성 하기

2. FUNCTION (내장 함수)

```
SQL> SELECT  DISTINCT deptno, count_mem(deptno)
  2  FROM emp;

    DEPTNO    COUNT_MEM(DEPTNO)
---------- ----------------------------
        30                    6
        20                    5
        10                    3
```

함수 사용 하기

2. FUNCTION (내장 함수)

함수 예제 3. 부서번호를 입력 받아 부서별로 평균 급여를 구해주는 함수

```
SQL> CREATE OR REPLACE FUNCTION avg_sal
  2   (s_deptno   emp.deptno%TYPE)
  3   RETURN   NUMBER
  4  IS
  5    avg_sal   NUMBER;
  6  BEGIN
  7    SELECT ROUND(AVG(sal),2) INTO avg_sal
  8    FROM emp
  9    WHERE deptno=s_deptno;
 10    RETURN avg_sal;
 11  END ;
 12  /

Function created.
```

함수 생성 하기

2. FUNCTION (내장 함수)

```
SQL> SELECT DISTINCT deptno, avg_sal(deptno)
  2  FROM emp;

    DEPTNO    AVG_SAL(DEPTNO)
----------  -- ---------------------
        30              1566.67
        10              2916.67
        20                 2175
```

함수 사용 하기

2. FUNCTION (내장 함수)

함수 예제 4. 사원번호를 입력 받아 해당 사원의 부서명을 알려주는 함수

```
SCOTT>CREATE OR REPLACE FUNCTION f_dname
  2  (v_empno  IN  emp.empno%TYPE)
  3  RETURN VARCHAR2
  4  IS
  5    v_dname  dept.dname%TYPE;
  6  BEGIN
  7    SELECT DNAME INTO V_DNAME
  8    FROM DEPT
  9    WHERE DEPTNO = (SELECT DEPTNO
 10                    FROM EMP
 11                    WHERE EMPNO=V_EMPNO);
 12    RETURN V_DNAME ;
 13  END ;
 14  /

Function created.
```

함수 생성 하기

2. FUNCTION (내장 함수)

```
SCOTT>SELECT ENAME, DEPTNO ,F_DNAME(EMPNO) "DNAME"
  2  FROM EMP ;

ENAME      DEPTNO    DNAME
---------- ---------- --------------------
홍길동             10    ACCOUNTING
일지매             30    SALES
SMITH              20    RESEARCH
ALLEN              30    SALES
WARD               30    SALES
JONES              20    RESEARCH
MARTIN             30    SALES
BLAKE              30    SALES
(이하 생략)
```

함수 사용 하기

2. FUNCTION (내장 함수)

함수 예제 5.

생성된 함수 조회하기

```
SCOTT>SELECT text
  2  FROM user_source
  3  WHERE type='FUNCTION'
  4  AND name='S_MAX_SAL' ;

TEXT
----------------------------------------
function s_max_sal
(s_deptno emp.deptno%type)
return number
is
  max_sal emp.sal%type ;
begin
  select max(sal) into max_sal
  from emp
  where deptno=s_deptno;
  return max_sal;
end;

11 rows selected.
```

3. ORACLE PACKAGE (패키지)

-패키지는 연관성이 높은 함수나 프로시저를 하나의 그룹으로 묶어두는 개념.

-패키지 선언부(Spec)와 패키지 몸체부(body)로 구성.

-패키지 선언부의 역할은 해당 패키지에 사용될 함수나 프로시저, 변수등에 대한 정의를 선언하는 부분

-패키지 몸체부에서는 선언부에서 선언된 함수나 프로시저등이 실제 구현되는 부분임.

-패키지 선언부에서 선언되지 않더라도 패키지 몸체부에서 사용될 수는 있지만 별로 권장사항은 아니니 가급적 선언부에서 선언하신 후 몸체부에서 사용해야 함

3. ORACLE PACKAGE (패키지)

1) PACKAGE 구조 : 선언부와 몸체부로 구성됨

- 패키지 선언부 생성

```
CREATE [OR REPLACE] PACKAGE package_name
IS | AS
   Public type and item declarations
   Subprogram specifications
END package_name ;
```

3. ORACLE PACKAGE (패키지)

- 패키지 몸체부(Package Body) 생성

```
CREATE [OR REPLACE] PACKAGE BODY package_name
  IS | AS
  Private type and item declarations
  Subprogram bodies
END package_name ;
```

-Subprogram bodies

이 부분이 실제 작동할 서브 프로그램(프로시저, 함수 등)을 기록하는 부분입니다. 단 주의해야 할 사항은 서브프로그램의 순서입니다. 기본적으로 참조되는 변수든 서브프로그램이든 참조하는 서브프로그램보다는 먼저 정의되어야 합니다. 일반적으로 PUBLIC 의 서브프로그램은 마지막 부분에 정의합니다

3. ORACLE PACKAGE (패키지)

2) 패키지 실행하기

패키지는 여러 환경에서 호출되어 실행될 수 있지만 생성된 패키지 오브젝트에 대한 실행권한을 가진 사용자만이 패키지를 호출하여 실행할 수 있습니다.

3) 패키지 삭제

패키지를 삭제할 때에는 패키지 선언부와 패키지 몸체부를 모두 삭제할 수 도 있고 패키지 몸체부만 삭제할 수도 있습니다.

```
DROP PACKAGE package_name ;
DROP PACKAGE BODY package_name ;
```

3. ORACLE PACKAGE (패키지)

4) Package 사용 예

예 1) Emp table 에서 총 급여합계와 평균 급여를 구하는 package 입니다

```
SQL> CREATE OR REPLACE PACKAGE emp_total
  2  AS
  3     PROCEDURE emp_sum;
  4     PROCEDURE emp_avg;
  5  END emp_total;
  6  /

Package created.
```

패키지 선언부 입니다.
Emp_sum , emp_avg 프로시져로
구성된 것을 볼 수 있습니다.

3. ORACLE PACKAGE (패키지)

```
SQL> CREATE OR REPLACE PACKAGE BODY emp_total  AS
  2  PROCEDURE emp_sum
  3  IS
  4    CURSOR emp_total_sum  IS
  5      SELECT COUNT(*), SUM(NVL(sal,0))
  6      FROM emp;
  7    total_num  NUMBER ;
  8    total_sum  NUMBER;
  9  BEGIN
 10    OPEN emp_total_sum ;
 11    FETCH emp_total_sum  INTO  total_num , total_sum ;
 12    DBMS_OUTPUT.PUT_LINE('총인원수: '||total_num||' , 급여합계: '||total_sum);
 13    CLOSE emp_total_sum;
 14  END emp_sum ; --emp_sum 프로시저 끝
```

패키지 몸체부 중
Emp_sum 프로시저 부분

다음 장에 계속....

3. ORACLE PACKAGE (패키지)

```
15  PROCEDURE emp_avg -- emp_avg 프로시저 시작
16  IS
17    CURSOR emp_total_avg  IS
18     SELECT  COUNT(*), AVG(NVL(sal,0))
19     FROM emp;
20    total_num NUMBER ;
21    total_avg  NUMBER ;
22  BEGIN
23    OPEN emp_total_avg ;
24    FETCH emp_total_avg  INTO  total_num , total_avg;
25      DBMS_OUTPUT.PUT_LINE('총인원수: '||total_num||' , 급여평균: '||total_avg);
26    CLOSE emp_total_avg ;
27   END emp_avg; -- 프로시저 끝
28  END emp_total; -- 패키지 끝
29  /

Package body created.
```

Emp_avg 프로시저 부분

3. ORACLE PACKAGE (패키지)

- 패키지 실행하기

SQL> SET SERVEROUTPUT ON
SQL> EXEC emp_total.emp_sum; -- 패키지 이름.프로시저 이름 으로 실행.

총인원수: 14 , 급여합계: 29025

PL/SQL procedure successfully completed.

SQL> EXEC emp_total.emp_avg;

총인원수: 14 , 급여평균: 2073.21428571428571428571

4. TRIGGER(트리거)

TRIGGER (트리거)

서브 프로그램 단위의 하나인 TRIGGER는 테이블, 뷰, 스키마 또는 데이터베이스에 관련된 PL/SQL 블록(또는 프로시저)으로 관련된 특정 사건(Event)이 발생될 때마다 묵시적(자동)으로 해당 PL/SQL 블록이 실행

TRIGGER는 데이터베이스 내에 오브젝트로서 저장되어 관리.
TRIGGER 자체는 사용자가 지정해서 실행을 할 수 없으며, 오직 TRIGGER 생성시 정의한 특정 사건(Event)에 의해서만 묵시적인 자동실행(Fire)

TRIGGER 를 생성하려면 CREATE TRIGGER, 수정하려면 ALTER TRIGGER, 삭제하려면 DROP TRIGGER 의 권한이 필요.
DATABASE 전체의 TRIGGER 조작은 ADMINISTER DATABASE TRIGGER 시스템 권한이 필요.
TRIGGER 에 대한 정보는 USER_OBJECTS, USER_TRIGGERS, USER_ERRORS 딕셔너리들을 조회하여 확인.
TRIGGER를 이루는 TRIGGER 몸체(실행부)에 TCL 명령, 즉 COMMIT, ROLLBACK, SAVEPOINT 명령이 포함될 수 없음

4. TRIGGER(트리거)

2) 주요 TRIGGER 유형

(1) 단순 DML TRIGGER

– **BEFORE TRIGGER**
테이블에서 **DML** 이벤트를 TRIGGER하기 전에 TRIGGER 본문을 실행

– **AFTER TRIGGER**
테이블에서 **DML** 이벤트를 TRIGGER한 후에 TRIGGER 본문을 실행.

- INSTEAD OF TRIGGER
TRIGGER 문 대신 TRIGGER 본문을 실행하며, 다른 방법으로는 수정이 불가능한 뷰에 사용

4. TRIGGER(트리거)

-**문장 TRIGGER / 행 TRIGGER**

문장 TRIGGER는 영향을 받는 행이 전혀 없더라도 TRIGGER가 한 번은 실행. 문장 TRIGGER는 TRIGGER 작업이 영향을 받는 행의 데이터 또는 TRIGGER 이벤트 자체에서 제공하는 데이터에 종속되지 않은 경우에 유용.

행 TRIGGER는 테이블이 TRIGGER 이벤트의 영향을 받을 때마다 실행되고, TRIGGER 이벤트의 영향을 받는 행이 없을 경우에는 실행되지 않음. 행 TRIGGER는 영향을 받는 행의 데이터나 TRIGGER 이벤트 자체에서 제공하는 데이터에 TRIGGER 작업이 종속될 경우에 유용.
행 TRIGGER 로 생성하려면 FOR EACH ROW 라는 구절을 사용.

4. TRIGGER(트리거)

(2) 혼합 TRIGGER (11g 부터 추가됨)

혼합 TRIGGER는 여러 가지 TRIGGER를 하나로 만든 것으로 마치 PL/SQL 의 패키지와 비슷

1. 혼합 TRIGGER는 **DML** TRIGGER여야 하며 테이블이나 뷰에 정의해야 함.
2. 혼합 TRIGGER의 본문은 **PL/SQL**에서 작성한 혼합 TRIGGER 블록이어야 함.
3. 혼합 TRIGGER 본문에는 초기화 블록이 포함될 수 없으므로 예외 섹션이 있을 수 없음.
4. 한 섹션에서 발생하는 예외는 해당 섹션에서 처리되어야 함.
   다른 섹션에서 처리하도록 권한을 이전할 수 없음.
5. **:OLD** 및 **:NEW**는 선언, **BEFORE STATEMENT** 또는 **AFTER STATEMENT** 섹션에 나타날 수 없음.
6. **BEFORE EACH ROW** 섹션만 **:NEW** 값을 변경할 수 있음.
7. **FOLLOWS** 절을 사용하지 않으면 혼합 TRIGGER의 실행 순서가 일정하지 않음.

4. TRIGGER(트리거)

(3) DML이 아닌 TRIGGER

- DDL 이벤트 TRIGGER
DML TRIGGER와 거의 동일하지만 TRIGGER 를 활용하여 DDL 작업을 하는 것만 다름.

– 데이터베이스 이벤트 TRIGGER
데이터베이스 이벤트 TRIGGER란 데이터베이스 내에서 생기는 일들을 관리하기 위해서 생성하는 TRIGGER . 사용자 관련 이벤트가 있고 시스템 관련 이벤트가 있으며 아래와 같음.

- 유저 이벤트 TRIGGER:
- 사용자가 발생시키는 작업에 TRIGGER를 생성.
- CREATE, ALTER 또는 DROP
- 로그온 또는 로그오프

- 데이터베이스 또는 시스템 이벤트 TRIGGER:
- 데이터베이스 전체에 영향을 주는 작업에 TRIGGER를 생성.
- 데이터베이스 종료 또는 시작
- 발생한 특정 오류 (또는 임의의 오류)

4. TRIGGER(트리거)

4) TRIGGER 생성

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
    event1 [ OR event2 OR event3 … ]
ON {table_name|view_name|SCHEMA|DATABASE}
[REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW [WHEN ( condition ) ] ]
    trigger_body
```

4. TRIGGER(트리거)

5) TRIGGER 관리

- 활성화/비활성화 하기
ALTER TRIGGER *trigger_name* DISABLE | ENABLE ;

- 특정 테이블에 속한 TRIGGER의 활성화/비활성화
ALTER TABLE *table_name* DISABLE | ENABLE ALL TRIGGERS ;

- **TRIGGER 수정 후 다시 컴파일하기**
ALTER TRIGGER *trigger_name* COMPILE ;

- TRIGGER 삭제
DROP TRIGGER *trigger_name* ;

- TRIGGER 조회하기
USER_TRIGGERS 를 조회하면 됩니다.

4. TRIGGER(트리거)

- TRIGGER 관련 권한들

- 스키마에서 TRIGGER를 생성, 변경 및 삭제할 수 있는 권한:
  - **GRANT CREATE TRIGGER TO SCOTT ;**
  - **GRANT ALTER ANY TRIGGER TO SCOTT;**
  - **GRANT DROP ANY TRIGGER TO SCOTT ;**

- 데이터베이스에서 TRIGGER를 생성할 수 있는 권한:
  - **GRANT ADMINISTER DATABASE TRIGGER TO SCOTT ;**

- **EXECUTE** 권한 **(**TRIGGER가 실행하는 스키마에 포함되지 않은 객체를 참조하는 경우**)**

4. TRIGGER(트리거)

- 예제1 : 테이블에 데이터를 입력할 수 있는 시간 지정하기(문장 레벨 Trigger 사용)

```
create table t_order
(no number,
ord_code varchar2(10),
ord_date date);
```

```
create or replace trigger t_order4
before insert on t_order4
begin
  if(to_char(sysdate,'HH24:MI') not between '11:05' and '11:10') then
    raise_application_error(-20100,'허용시간이 아닙니다');
  end if;
end;
/
```

```
insert into t_order4 values(1, 'c100', sysdate);
```

4. TRIGGER(트리거)

- 예제2: 테이블에 데이터 값을 지정하고 그 외의 값이 입력되면 에러를 발생시키는 TRIGGER 생성

```
create or replace trigger t_order2
before insert on t_order4
for each row
begin
  if(:new.ord_code) not in('c100', 'c200') then
    raise_application_error(-20200,'제품코드가 틀립니다');
  end if;
end;
/

insert into t_order4 values(3, 'c300', sysdate);
```

4. TRIGGER(트리거)

- 예제3: TRIGGER 작동 조건을 when절로 정의

```
create or replace trigger t_order6
before INSERT on t_order4
  for each row
  when (new.ord_code= 'c500')
begin
  if(to_char(sysdate,'HH24:MI') not between '11:40' and '11:50') then
    raise_application_error(-20300, 'c500 제품 입력시간이 아닙니다');
  end if;
end;
/
insert into t_order4 values(4, 'c500', sysdate);
```

4. TRIGGER(트리거)

- 예제4: 기존 테이블(t_test1)의 내용을 업데이트할때 기존의 내용을 백업용 테이블에(t_test2)로 옮겨 놓는 TRIGGER 작성

```
create table t_test1
(no number, name varchar2(10));

create table t_test2
as select * from t_test1;
```

```
insert into t_test1 values(1, 'AAA');
insert into t_test1 values(2, 'BBB');
insert into t_test1 values(3, 'CCC');
```

```
create or replace trigger t_move
before update on t_test1
for each row
begin
  insert into t_test2 values(:old.no, :old.name);
end;
/
```

```
update t_test1 set name='A1A2' where no=1;
select * from t_test1;
select * from t_test2;
```

- 기본 테이블(t_test1)테이블의 데이터가 삭제될 때 백업 테이블(t_test3)로 이동시키며 백업테이블에 삭제한 시간 삭제 전 데이터를 모두 기록하는 trigger 생성

```
create table t_test4(no number, name varchar2(10), time date);

create or replace trigger d_trigger
after delete on t_test1
for each row
begin
  insert into t_test3 values(:old.no, :old.name, sysdate);
end;
/
delete from t_test1 where no=1;
select * from t_test3;
```

4. TRIGGER(트리거)

상품 테이블

품번	항목명	단가
100	새우깡	900
200	감자깡	900
300	맛동산	1000

입고 테이블

품번	수량	금액
100	10	9000
200	10	9000
300	10	10000

판매 테이블

품번	수량	금액
100	0	0
200	0	0
300	0	0

재고 테이블

품번	수량	금액
100	10	9000
200	10	9000
300	10	10000

4. TRIGGER(트리거)

연습 문제 1)
아래의 입고 SQL 과 같이 새우깡이 입고되면 재고 테이블에서 자동으로 새우깡의 재고 수량과 금액이 증가 되는 TRIGGER를 작성하세요.

입고 SQL : SQL> INSERT INTO 입고 VALUES (100,2,1800);

연습문제 2)
위 테이블에서 새우깡이 판매되면 재고 테이블에서 새우깡의 재고 수량과 금액에 반영되도록 감소하는 TRIGGER를 작성하세요.

판매 SQL: SQL> INSERT INTO 판매 VALUES (100,3,2700);