

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

8주차: PL/SQL Module(Stored Block3)

I. 시작

[학습목표]

8차시에는 Stored Block을 활용하여 사용자 정의 Package,Trigger를 학습합니다.

[학습목차]

8-1 PACKAGE

8-2 TRIGGER

II. 중간

8-1 PACKAGE

PACKAGE 는 사전적으로 꾸러미,소포라는 의미를 가지고 있습니다.

꾸러미는 어떤 물건을 몇 개씩 꾸러 모아서 한뭉치로 만든 것을 의미 합니다.

PL/SQL 의 PACKAGE 는 무엇을 꾸러 모았을까요?

PROCEDURE,FUNCTION,CURSOR,VARIABLE를 꾸러미(보자기)로 한뭉치로 모아 놓은것으로 PL/SQL의 PACKAGE는 연관된 PROCEDURE,FUNCTION,VARIABLE, CURSOR을 함께 묶어서 만든 OBJECT 이다.

8-1-1 PACKAGE 구성

PACKAGE는 2 영역으로 구성 됩니다.

(1) PACKAGE HEADER 영역

(2) PACKAGE BODY 영역

PACKAGE HEADER 영역은 PACKAGE에 포함된 내용에 대한 명세(SPECIFICATION)서 입니다. 즉 PACKAGE내에 포함된 PROCEDURE/FUNCTION등에 대한 이름/입출력 정의를 선언하는 영역으로 구체적인 소스코드를 가지고 있지는 않습니다.

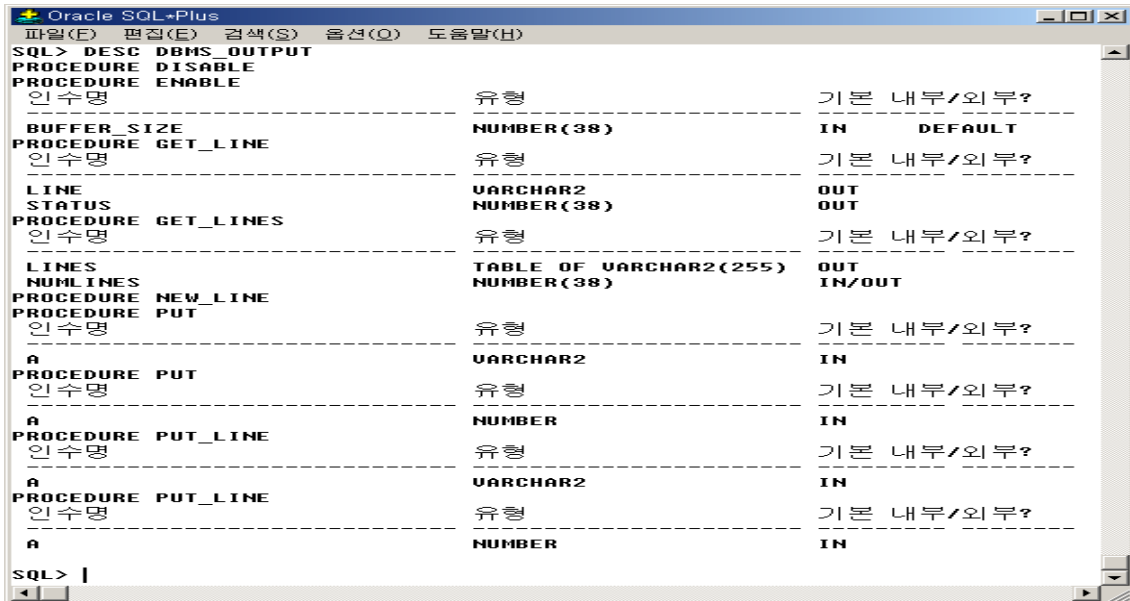
PACKAGE BODY 영역은 HEADER 영역에서 정의한 명세에 대한 구체적인 처리 LOGIC을 가지게 됩니다. 즉 구체적인 소스 코드를 가지고 있습니다.

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

- < 참고> PACKAGE는 (1) Oracle社에서 만들어 제공하는 PACKAGE
(2) 사용자 정의 PACKAGE

DESC 명령어를 통해 PROCEDURE/FUNCTION의 입/출력 구성의 정의를 보았는데 PACKAGE 역시 DESC 명령어로 조회가 가능합니다.

그동안 우리가 사용해왔던 DBMS_OUTPUT 이라는 Oracle社 제공 패키지를 봅시다.



인수명	유형	기본 내부/외부?
BUFFER_SIZE	NUMBER(38)	IN DEFAULT
PROCEDURE GET_LINE		
인수명	유형	기본 내부/외부?
LINE	VARCHAR2	OUT
STATUS	NUMBER(38)	OUT
PROCEDURE GET_LINES		
인수명	유형	기본 내부/외부?
LINES	TABLE OF VARCHAR2(255)	OUT
NUMLINES	NUMBER(38)	IN/OUT
PROCEDURE NEW_LINE		
PROCEDURE PUT		
인수명	유형	기본 내부/외부?
A	VARCHAR2	IN
PROCEDURE PUT		
인수명	유형	기본 내부/외부?
A	NUMBER	IN
PROCEDURE PUT_LINE		
인수명	유형	기본 내부/외부?
A	VARCHAR2	IN
PROCEDURE PUT_LINE		
인수명	유형	기본 내부/외부?
A	NUMBER	IN

자세히 보면

- (1) 위의 PACKAGE는 9개의 PROCEDURE로 구성
- (2) 각각의 PROCEDURE의 입/출력 파라미터의 정의

EX) PUT_LINE은 함수 오버로딩(OVERLOADING)으로 정의 되어 있으며
1개의 파라미터를 가지고 있습니다. 파라미터의 데이터
타입(VARCHAR2/NUMBER) 과 개수에 따라 개별 동작을 수행

[질문] 지금 보이는 내용은 PACKAGE HEADER 일까요 PACKAGE BODY일까요?
PACKAGE HEADER 영역에 정의된 명세(SPECIFICATION) 입니다.
명세(SPECIFICATION)에서는 PROCEDURE/FUNCTION/VARIABLE 를 정의
하지만 PROCEDURE/FUNCTION 만이 DESC를 통해 조회 가능

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

8-1-2 PACKAGE 실습

8_PACKAGE_1.SQL

[참고] PACKAGE의 HEADER(명세)만 선언하여 SESSION별 전역 변수로 사용하는 예

```
CREATE OR REPLACE PACKAGE P_GLOBAL_VARS
AS
    -- 명세(HEADER, SPECIFICATION)에서 선언하는 경우 PUBLIC하게 사용.
    LAST_CHANGE_DATE    DATE;
    MAX_VALUE            NUMBER(4);
END;
/

DESC P_GLOBAL_VARS

SET SERVEROUTPUT ON

-- 첫번째 BLOCK에서 변수값 초기화
BEGIN
    P_GLOBAL_VARS.MAX_VALUE := 3000;
    P_GLOBAL_VARS.LAST_CHANGE_DATE := SYSDATE;
    DBMS_OUTPUT.PUT_LINE('BLOCK1 P_GLOBAL_VARS.MAX_VALUE = '||P_GLOBAL_VARS.MAX_VALUE);
END;
/

-- 서로 다른 독립적인 BLOCK간에 DATA 공유
BEGIN
    P_GLOBAL_VARS.MAX_VALUE := P_GLOBAL_VARS.MAX_VALUE + 3000;
    DBMS_OUTPUT.PUT_LINE('BLOCK2 P_GLOBAL_VARS.MAX_VALUE = '||P_GLOBAL_VARS.MAX_VALUE);
    DBMS_OUTPUT.PUT_LINE('BLOCK2 P_GLOBAL_VARS.LAST_CHANGE_DATE = '||
        TO_CHAR(P_GLOBAL_VARS.LAST_CHANGE_DATE, 'YYYY-MM-DD'));
END;
/
```

- ① OR REPLACE는 OPTION이지만 습관적으로 사용하는 것이 편리 하며
P_GLOBAL_VARS는 PACKAGE의 이름, PACKAGE는 HEADER 와 BODY로
구성되지만 위의 예제는 HEADER 부분만 정의된 예제
HEADER 의 선언부를 관찰해보면 실행부(BEGIN ~ END)가 없다. 왜?
HEADER 부분에서는 PACKAGE내에서 사용되는 PROCEDURE/FUNCTION/VARIABLE의
명세(SPECIFICATION)을 정의하는 부분이기 때문에 선언부(AS ~ END)만이 있다.
선언부(AS ~ END)에서 2개의 변수를 선언 합니다.
- ② HEADER 영역에서 정의한 변수는 DESC를 통해서 볼 수 없다.
- ③ 첫번째 BLOCK에서 두변수의 값을 초기화 합니다.
PACKAGE내에 정의된 FUNCTION/PROCEDURE/VARIABLE을 사용하기
위해서는 PACKAGE명.OBJECT명(EX P_GLOBAL_VARS.MAX_VALUE)표기 사용
- ④ 두번째 BLOCK에서
P_GLOBAL_VARS.MAX_VALUE := P_GLOBAL_VARS.MAX_VALUE + 3000;

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

P_GLOBAL_VARS.MAX_VALUE에는 어떤값이 연산의 결과로 저장될까?

2개의 DBMS_OUTPUT에 의해 출력된 결과는 이전 BLOCK에서 초기화된 값을

새로운 두번째 BLOCK에서 참조하여 사용할수 있다는 결과를 보여 줍니다.

위 실습은 PACKAGE의 HEADER 영역에 변수를 정의하여 ORACLE DBMS의 SESSION내에 GLOBAL 변수로 활용할수 있는 예제 입니다.

현업에서 PACKAGE를 활용할 때 위의 예제 처럼 사용하는 경우는 거의 없습니다.

대부분은 DBMS내의 데이터를 핸들링하는 용도로 사용합니다.

8-1-3 PACKAGE 실습

이전 8-1-2 에서는 PACKAGE의 간단한 사용을 보았습니다.

현업에서 PACKAGE를 사용하는 대부분의 이유는 데이터 핸들링이겠죠!

데이터를 핸들링하는 PACKAGE에 대해 실습을 해봅시다.

8_PACKAGE_2_HEADER.SQL

```
CREATE OR REPLACE PACKAGE P_EMPLOYEE
AS
    -----
    -- PUBLIC PROCEDURE/FUNCTION/VARIABLE 명세 (=SPECIFICATION)
    -----

    -- 사원 퇴사 처리
    PROCEDURE DELETE_EMP(P_EMPNO EMP.EMPNO%TYPE);

    -- 신규 입사사원 등록
    PROCEDURE INSERT_EMP(P_EMPNO NUMBER, P_ENAME VARCHAR2, P_JOB VARCHAR2, P_SAL NUMBER, P_DEPTNO NUMBER);

    -- 해당사원의 MANAGER의 이름을 RETURN하는 함수
    FUNCTION SEARCH_MNG(P_EMPNO EMP.EMPNO%TYPE) RETURN VARCHAR2;

    GV_ROWS          NUMBER(6);                                -- PUBLIC 변수
END P_EMPLOYEE;
/
```

① 2개의 PROCEDURE 와 1개의 FUNCTION 1개의 변수를 묶은 PACKAGE HEADER 입니다.

객체지향언어의 CLASS 개념에서 PUBLIC 과 PRIVATE를 잠시 생각해봅시다.

객체지향언어에서 PUBLIC은 CLASS 외부에 공개된 METHOD 이고

PRIVATE은 CLASS 외부에 공개되지 않은 METHOD 입니다.

Oracle PACKAGE에서도 PUBLIC 과 PRIVATE 개념이 있습니다.

HEADER 영역에서 정의한 PROCEDURE/FUNCTION/VARIABLE은

접근 지정자 PUBLIC 으로 PACKAGE 외부에 공개된 METHOD.

공개가 되었다는 의미는 호출하여 사용할수 있다는 의미입니다.

예를 들면 P_EMPLOYEE.DELETE_EMP(7369);

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

BODY 영역에서만 정의한 PROCEDURE/FUNCTION/VARIABLE은 접근 지정자 PRIVATE 입니다. 외부에서 사용자가 직접 호출하여 사용할수 없습니다.

8_PACKAGE_2_BODY.SQL

```
-- PACKAGE BODY : 실제 처리 LOGIC을 정의하는 부분
CREATE OR REPLACE PACKAGE BODY P_EMPLOYEE
AS
    V_ENAME          EMP.ENAME%TYPE;          -- PRIVATE 변수
    V_ROWS           NUMBER(6);                -- PRIVATE 변수

    FUNCTION PRVT_FUNC(P_NUM IN NUMBER) RETURN NUMBER IS -- PRIVATE FUNCTION 정의
    BEGIN
        RETURN P_NUM;
    END PRVT_FUNC;

    PROCEDURE INSERT_EMP(P_EMPNO NUMBER,P_ENAME VARCHAR2,P_JOB VARCHAR2,P_SAL NUMBER,P_DEPTNO NUMBER)
    IS
    BEGIN
        INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DEPTNO) VALUES(P_EMPNO,P_ENAME,P_JOB,P_SAL,P_DEPTNO);
        COMMIT;
    END INSERT_EMP;

    PROCEDURE DELETE_EMP(P_EMPNO EMP.EMPNO%TYPE) IS          -- PUBLIC PROCEDURE 정의
    BEGIN
        DELETE FROM EMP WHERE EMPNO = P_EMPNO;
        COMMIT;
        --IMPLICIT CURSOR ATTRIBUTE,PUBLIC변수 참조
        GV_ROWS := GV_ROWS + SQL%ROWCOUNT;
        V_ROWS := PRVT_FUNC(GV_ROWS);                -- PRIVATE FUNCTION 참조
    EXCEPTION
        WHEN OTHERS THEN
            ROLLBACK;
            WRITE_LOG('P_EMPLOYEE.DELETE',SQLERRM,'VALUES : [EMPNO]=>'||P_EMPNO);
    END DELETE_EMP;

    FUNCTION SEARCH_MNG(P_EMPNO EMP.EMPNO%TYPE) RETURN VARCHAR2
    IS
    BEGIN
        V_ENAME          EMP.ENAME%TYPE;
        -- 사원의 매니저 이름 검색
        SELECT ENAME INTO V_ENAME FROM EMP
        WHERE EMPNO = (SELECT MGR FROM EMP WHERE EMPNO = P_EMPNO);

        RETURN V_ENAME;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            V_ENAME := 'NO_DATA';
            RETURN V_ENAME;
        WHEN OTHERS THEN
            V_ENAME := SUBSTR(SQLERRM,1,12);
            RETURN V_ENAME;
    END SEARCH_MNG;

BEGIN
    -- -----
    -- PROCEDURE/FUNCTION 와는 달리 PACKAGE에서는 BEGIN 실행부가 OPTIONAL
    -- -----
    GV_ROWS := 0;                                -- 주로 초기화 작업 수행하는 역할..
END P_EMPLOYEE;
/
```

① PACKAGE BODY 영역에서 정의하는 변수는 PRIVATE 변수에 해당 하며.

변수 Scope(영역)은 PACKAGE 내에서는 전역 변수로 사용됨

참조는 PACKAGE 내부에 정의된 PROCEDURE/FUNCTION/VARIABLE들만 참조가 가능
외부에서 직접적인 참조는 불가능하다.

PUBLIC 변수는 사용자가 직접 참조가 가능 GV_ROWS는 참조 가능 하지만 V_ROWS는
에러가 발생한다.'

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

```
SQL> BEGIN
  2   P_EMPLOYEE.GV_ROWS  := 100;
  3   P_EMPLOYEE.V_ROWS   := 100;
  4   END;
  5   /
P_EMPLOYEE.V_ROWS      := 100;
      *
```

3행에 오류:

ORA-06550: 줄 3, 열 13:PLS-00302: 'V_ROWS' 구성 요소가 정의되어야 합니다
ORA-06550: 줄 3, 열 2:PL/SQL: Statement ignored

①-② PRIVATE 변수는 PACKAGE 내부 구성 요소(PROCEDURE/FUNCTION/VARIABLE)가 참조 가능

④ 단락을 보면 PROCEDURE 내에서 참조를 하여 사용

```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, LAST_NAME, JOB_ID, SALARY) VALUES (1001, 'SCOTT', 'ANALYST', 3000);
GV_ROWS := GV_ROWS + SQL%ROWCOUNT;
V_ROWS := PVT_FUNC(GV_ROWS);
```

② 입력받은 값을 그대로 리턴해주는 무의미한 기능을 하는 FUNCTION으로 PRIVATE FUNCTION의 예제를 보여 드리기 위한 말그대로 예제를 위한 예제 입니다.

[질문] 왜 PRIVATE FUNCTION 인가?

③ 사번,이름,직군,급여등을 입력받아 사원(EMP) 테이블 INSERT 하는 PROCEDURE 입니다. 실행부만 명시하고 예외 처리부는 없는 간단한 PROCEDURE 이죠!
실무에서는 이렇게 개발하면?

④ 사번을 넣으면 해당 사원의 정보를 삭제하는 DELETE_EMP PROCEDURE 입니다.

- 파라미터 정의시 참조 연산자(%TYPE) 사용이 가능하죠!
- SQL%ROWCOUNT(INSERT,DELETE,UPDATE로)처리된, SELECT로 추출된 로우 건수를 지시하는 커서 속성자(암시적 커서)를 사용하여 삭제된 ROW의 개수 저장
- PUBLIC 변수를 참조하는 예제 와 PACKAGE내의 PRIVATE 변수를 참조하는 예제
- 예외 처리부를 가지고 있으며

예외 발생시 이전 TRANSACTION을 ROLLBACK 처리하고

WRITE_LOG를 호출하여 발생한 에러를 기록(LOG)한다.

⑤ 사번을 입력 받아 해당 사원의 매니저 이름을 리턴하는 함수

- 함수내에 LOCAL 변수를 정의
- SELECT 구문 사용시 INTO 이하의 변수에 해당 값 저장
- SQLERRM 을 참조하여 발생한 에러 메시지를 함수의결과로리턴(예제를 위한 예제)

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

⑥ PACKAGE의 실행부.

PROCEDURE/FUNCTION에서는 실행부(BEGIN ~ END)는 필수적 영역 이지만

PACKAGE에서 실행부(BEGIN ~ END)는 선택적 영역 입니다.

PACKAGE에서 실행부의 주된 용도는 PACKAGE내부의 값을 초기화 하는 용도 입니다.

예제상에서는 PUBLIC 변수인 GV_ROWS 값을 초기화 합니다.

```
DESC P_EMPLOYEE
BEGIN
    P_EMPLOYEE.GV_ROWS := 100; -- PUBLIC 변수 참조 => 사용자 직접 참조 가능
    P_EMPLOYEE.V_ROWS := 100; -- PRIVATE 변수 참조 => 사용자 직접 참조 불가능
END;
/

BEGIN
    P_EMPLOYEE.INSERT_EMP(1111,'PACKAGE', 'CIO',9999,10);
    P_EMPLOYEE.INSERT_EMP(1112,'PACKAGE2','CTO',9999,20);
    P_EMPLOYEE.DELETE_EMP(1112);
    DBMS_OUTPUT.PUT_LINE('DELETED ROWS => '||P_EMPLOYEE.GV_ROWS);
END;
/

DECLARE
    V_ENAME EMP.ENAME%TYPE;
BEGIN
    V_ENAME := P_EMPLOYEE.SEARCH_MNG(1111);
    DBMS_OUTPUT.PUT_LINE('MANAGER NAME => '||V_ENAME);
END;
/
```

① DESC P_EMPLOYEE

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

DESC 명령어로 PACKAGE의 구성요소들을 알수 있다.

[질문] PACKAGE내의 모든 구성요소들이 표시되는가?

②

```
P_EMPLOYEE.GV_ROWS := 100;      -- PUBLIC 변수 참조 => 사용자 직접 참조 가능
P_EMPLOYEE.V_ROWS  := 100;      -- PRIVATE 변수 참조 => 사용자 직접 참조 불가능
실행예제에서 PRIVATE변수를 참조 하려다 에러발생
```

- ③ P_EMPLOYEE.INSERT_EMP 를 사용하여 2개의 ROW를 INSERT
P_EMPLOYEE.DELETE_EMP 를 사용하여 1개의 ROW를 DELETE
P_EMPLOYEE.GV_ROWS 와 DBMS_OUTPUT을 사용하여 삭제된 ROW의
개수 출력. DELETED ROWS => 0 실행결과가 정상적인지?

우리가 예상한 결과는 DELETED ROWS => 1인데... 실제 삭제 되었지만 이유가
무엇일까?

<8_PACKAGE_2.SQL 참조> PACKAGE 소스를 보면

```
PROCEDURE DELETE_EMP(P_EMPNO EMP.EMPNO%TYPE) IS      -- PUBLIC PROCEDURE 정의
BEGIN
    DELETE FROM EMP WHERE EMPNO = P_EMPNO;
    COMMIT;
    --IMPLICIT CURSOR ATTRIBUTE,PUBLIC변수 참조
    GV_ROWS := GV_ROWS + SQL%ROWCOUNT;
    V_ROWS  := PRVT_FUNC(GV_ROWS);                  -- PRIVATE FUNCTION 참조
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        WRITE_LOG('P_EMPLOYEE.DELETE',SQLERRM,'VALUES : [EMPNO]=>'||P_EMPNO);
END DELETE_EMP;
```

DELETE → COMMIT → SQL%ROWCOUNT 순서를

DELETE → SQL%ROWCOUNT → COMMIT 으로 바꾸어 주어야만 합니다.

이유는 무엇일까?

커서 속성자는 가장 최근 실행된 SQL의 속성을 참조 한다

따라서 위의 예제는 COMMIT 실행을 참조하는 결과

- ④ V_ENAME := P_EMPLOYEE.SEARCH_MNG(1111);

1111 사원의 매니저 이름을 찾아오는 함수를 실행한 결과 'NO_DATA'가 리턴
되었습니다.

PACKAGE 소스에서 SEARCH_MNG를 보면

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

```
FUNCTION SEARCH_MNG(P_EMPNO EMP.EMPNO%TYPE) RETURN VARCHAR2
IS
    V_ENAME EMP.ENAME%TYPE;
BEGIN
    -- 사원의 매니저 이름 검색
    SELECT ENAME INTO V_ENAME FROM EMP
    WHERE EMPNO = (SELECT MGR FROM EMP WHERE EMPNO = P_EMPNO);

    RETURN V_ENAME;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        V_ENAME := 'NO_DATA';
        RETURN V_ENAME;
    WHEN OTHERS THEN
        V_ENAME := SUBSTR(SQLERRM,1,12);
        RETURN V_ENAME;
END SEARCH_MNG;
```

1111 사원의 매니저가 없는 경우 해당 SELECT는 NO_DATA_FOUND 라는 EXCEPTION을 발생한다. NO_DATA_FOUND 예외처리부에서 NO_DATA 를 리턴하도록 합니다.

<참고> P_EMPLOYEE PACKAGE를 구성하는 2개의 FUNCTION과 2개의 PROCEDURE를 작성하였다. 이중에 1개의 PROCEDURE 와 FUNCTION에서만 예외처리를 했지만.. 실제 개발시에는 ?

- < 참고> PACKAGE를 학습하면서 객체지향언어의 CLASS 개념과 유사성
- 함수 오버로딩(OVERLOADING)
 - 연관된 Method 를 CLASS 내에 정의
 - PUBLIC 과 PRIVATE 영역 등의 특징들이...

PL/SQL은 ADA(에이다)라는 언어를 모델로 만들어진 언어 이다..

ADA(에이다) 언어의 주요특징중 하나는 객체지향 프로그래밍을 지원하는 부분입니다.

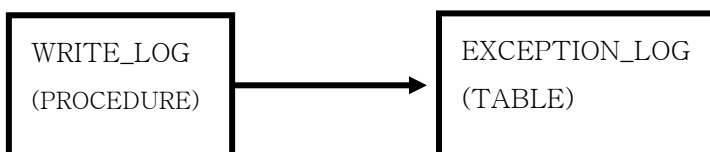
8-2 종속 관계

Stored Block(FUNCTION/PROCEDURE/PACKAGE)으로 개발시 OBJECT간의 종속관계를 이해해야 한다.

종속성이라는 단어를 참조라는 의미로 이해하면 접근하기가 쉽습니다.

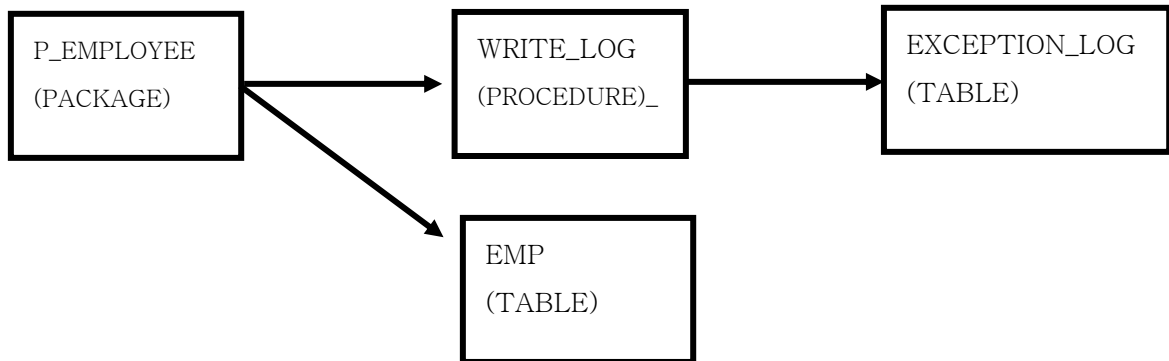
WRITE_LOG PROCEDURE는 EXCEPTION_LOG 테이블에 INSERT(참조) 합니다.

WRITE_LOG → EXCEPTION_LOG



데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

P_EMPLOYEE PACKAGE를 보면 아래의 참조 관계를 가지고 있습니다.



참조관계를 알아야 하는 이유가 무엇일까요?

예를들어 EXCEPTION_LOG 테이블에 문제가 발생하여 사용할수 없거나 문제가 생기면 해당 테이블을 참조하는 모든 PROCEDURE/FUNCTION/PACKAGE에도 영향을 미치게 된다.

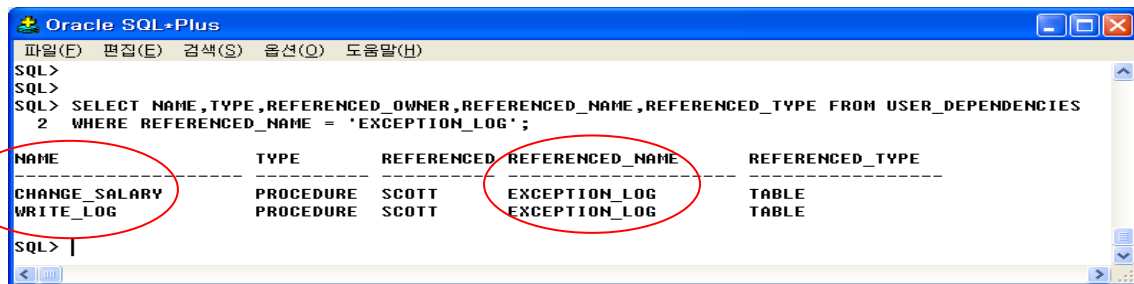
EXCEPTION_LOG 테이블의 구조에 변경(컬럼 추가)을 발생 시킵니다.

```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> ALTER TABLE EXCEPTION_LOG ADD(DESCRIPTION2 VARCHAR2(250));
테이블이 변경되었습니다.
SQL> DESC EXCEPTION_LOG
이름          널?          유형
-----
LOG_DATE      VARCHAR2(8)
LOG_TIME      VARCHAR2(6)
PROGRAM_NAME  VARCHAR2(100)
ERROR_MESSAGE VARCHAR2(250)
DESCRIPTION   VARCHAR2(250)
DESCRIPTION2   VARCHAR2(250)
SQL>
```

해당 테이블을 참조하는 OBJECT들은 어떤 상태가 될까?

```
SELECT NAME,TYPE,REFERENCED_OWNER,REFERENCED_NAME,REFERENCED_TYPE
FROM USER_DEPENDENCIES
WHERE REFERENCED_NAME = 'EXCEPTION_LOG';
```

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)



Oracle SQL*Plus

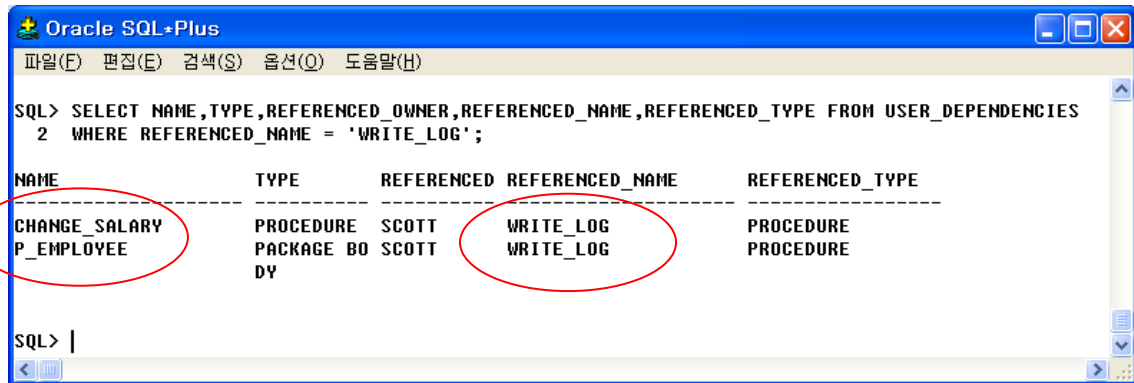
```
SQL> SELECT NAME,TYPE,REFERENCED_OWNER,REFERENCED_NAME,REFERENCED_TYPE FROM USER_DEPENDENCIES
2 WHERE REFERENCED_NAME = 'EXCEPTION_LOG';
```

NAME	TYPE	REFERENCED_OWNER	REFERENCED_NAME	REFERENCED_TYPE
CHANGE_SALARY	PROCEDURE	SCOTT	EXCEPTION_LOG	TABLE
WRITE_LOG	PROCEDURE	SCOTT	EXCEPTION_LOG	TABLE

SQL> |

위의 결과를 보면 EXCEPTION_LOG 테이블을 2개의 PROCEDURE인 CHANGE_SALARY와 WRITE_LOG 가 참조하고 있다. WRITE_LOG 를 참조하는 OBJECT를 찾아 보시다.

```
SELECT NAME,TYPE,REFERENCED_OWNER,REFERENCED_NAME,REFERENCED_TYPE
FROM USER_DEPENDENCIES
WHERE REFERENCED_NAME = 'WRITE_LOG';
```



Oracle SQL*Plus

```
SQL> SELECT NAME,TYPE,REFERENCED_OWNER,REFERENCED_NAME,REFERENCED_TYPE FROM USER_DEPENDENCIES
2 WHERE REFERENCED_NAME = 'WRITE_LOG';
```

NAME	TYPE	REFERENCED_OWNER	REFERENCED_NAME	REFERENCED_TYPE
CHANGE_SALARY	PROCEDURE	SCOTT	WRITE_LOG	PROCEDURE
P_EMPLOYEE	PACKAGE BODY	SCOTT	WRITE_LOG	PROCEDURE

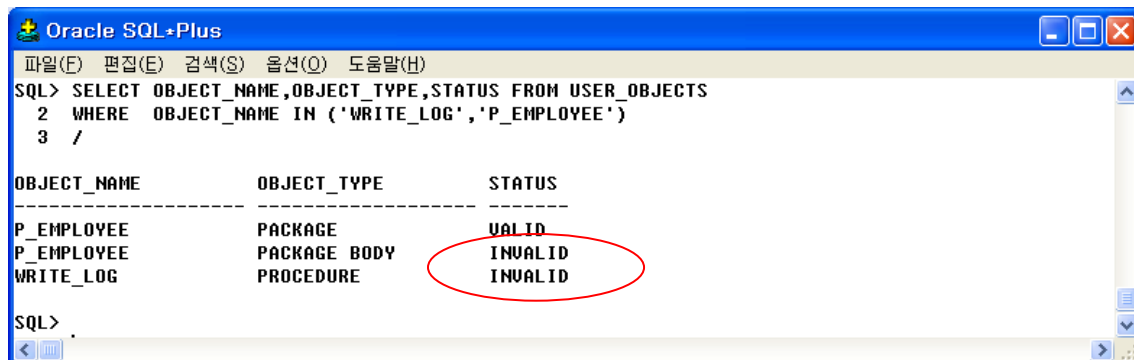
SQL> |

참조 관계를 그려 보시다

P_EMPLOYEE → WRITE_LOG → EXCEPTION_LOG

참조 되고 있는 OBJECT에 변경이 발생되 관련된 OBJECT의 상태 변화를 조회해보면..

```
SELECT OBJECT_NAME,OBJECT_TYPE,STATUS FROM USER_OBJECTS
WHERE OBJECT_NAME IN ('WRITE_LOG','P_EMPLOYEE');
```



Oracle SQL*Plus

```
SQL> SELECT OBJECT_NAME,OBJECT_TYPE,STATUS FROM USER_OBJECTS
2 WHERE OBJECT_NAME IN ('WRITE_LOG','P_EMPLOYEE')
3 /
```

OBJECT_NAME	OBJECT_TYPE	STATUS
P_EMPLOYEE	PACKAGE	VALID
P_EMPLOYEE	PACKAGE BODY	INVALID
WRITE_LOG	PROCEDURE	INVALID

SQL> .

참조하는 OBJECT의 상태가 INVALID의 상태 입니다. INVALID는 사용할수 없는 상태를 의미 합니다. <참고> OBJECT_TYPE중 PACKAGE는 PACKAGE Header 의미

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

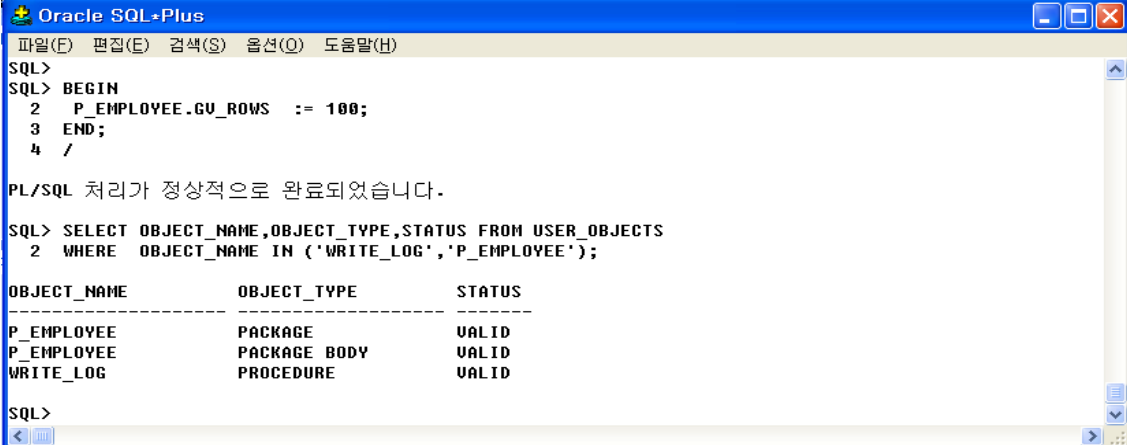
PACKAGE Header 가 VALID인 이유는 실제적인 참조가 아니라 PUBLIC 영역에 대한 정의만 하기 때문

<참고> 실무에서 복잡한 Stored Block(FUNCTION/PROCEDURE/PACKAGE)을 활용하여 프로젝트를 진행한다면 참조 관계에 대한 이해를 가지고 있어야 하며 참조 관계를 SQL을 사용하여 조회 할수 있어야 함,

위의 예제에서 참조되는 OBJECT 와 참조하는 OBJECT간의 상관 관계를 보았습니다. 참조 되는 OBJECT(EXCEPTION_LOG)의 구조가 변경되어 참조하는 OBJECT들이 일시적으로 VALID → INVALID로 상태가 변경되었지만 참조하는 OBJECT를 재실행하면 PL/SQL 엔진이 자동으로 재 컴파일을 하여 해당 OBJECT의 상태가 VALID로 변경되면서 실행 됩니다.

위의 예제와 달리 참조되는 OBJECT가 지속적으로 사용할수 없는 상태가 된다면 참조하는 모든 OBJECT는 사용할수 없는(INVALID)가 됩니다. 복잡한 참조관계를 가지는 경우에는 참조 관계를 조회 하는 방법을 이해해야 합니다.

자~ P_EMPLOYEE를 실행하면 어떤일이 발생하는지 봅시다.



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> BEGIN
  2   P_EMPLOYEE.GU_ROWS  := 100;
  3   END;
  4   /

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> SELECT OBJECT_NAME,OBJECT_TYPE,STATUS FROM USER_OBJECTS
  2   WHERE OBJECT_NAME IN ('WRITE_LOG','P_EMPLOYEE');

OBJECT_NAME      OBJECT_TYPE      STATUS
-----
P_EMPLOYEE       PACKAGE          VALID
P_EMPLOYEE       PACKAGE BODY     VALID
WRITE_LOG        PROCEDURE        VALID

SQL>
```

PACKAGE를 실행합니다. PL/SQL 엔진이 관련 PACKAGE를 재 컴파일 합니다. 컴파일 하는 과정에서 참조되는 OBJECT의 유효성을 검사하여 유효하다면(VALID)하다면 참조하는 OBJECT의 상태를 VALID로 만들고나서 실행합니다. OBJECT의 STATUS (VALID/INVALID)를 조회해보면 INVALID → VALID로 변경됨을 알수 있다.

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

8-2 TRIGGER

8-2-1 TRIGGER 개요 실습

TRIGGER의 사전적으로 명사적으로는 “방아쇠”라는 의미

동사적으로는 “시작하게 하다”는 의미.

자동으로 방아쇠를 당겨 무엇인가를 시작하게 한다는 것을 연상을 하면

TRIGGER를 쉽게 이해할수 있습니다.

간단한 TRIGGER를 먼저 만들어 봅시다.

우리회사의 급여의 최고 한계가 9000 이어서 급여는 9000이상 지급될수 없지만 개발되어 있는 급여 관리 프로그램을 통해 9000이상의 급여로 종종 수정이 되곤 합니다. 물론 프로그램의 기능을 수정할수 있지만 TRIGGER를 사용하여 문제를 해결 해봅시다.

아래의.TRIGGER는 사원의 급여가 9000이상으로 수정(UPDATE) 되지 못하도록 하는 기능을 합니다.

<8_TRIGGER_1.SQL>

```
CREATE OR REPLACE TRIGGER TRG_CHANGE_SAL
BEFORE UPDATE OF SAL ON EMP
FOR EACH ROW
BEGIN
    IF ( :NEW.SAL > 9000 ) THEN
        :NEW.SAL := 9000;
    END IF;
END;
```

① CREATE OR REPLACE는 이전과 동일 의미 입니다.

TRG_CHANGE_SAL은 TRIGGER의 이름 입니다.

TRIGGER는 이벤트, 시기,횟수, 대상을 주요하게 이해해야 합니다.

동작하는 이벤트에는 INSERT,DELETE.UPDATE가 있습니다.

동작하는 시기는 BEFORE 와 AFTER가 있습니다.

동작하는 LEVEL은 ROW LEVEL 과 STATEMENT LEVEL이 있습니다.

동작하는 이벤트는 예제를 기준으로 보면 EMP 테이블의 SAL 컬럼에

UPDATE 를 수행할 때 TRIGGER가 시작된다는 의미

동작하는 시기는 예제를 기준으로 보면 BEFORE는 UPDATE 와 TRIGGER중

UPDATE 전에 TRIGGER가 수행된다는 의미

AFTER는 TRIGGER 이벤트가 수행된후 TRIGGER가

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

동작한다는 의미 입니다.

동작하는 LEVEL은 TRIGGER가 실행되는 회수를 의미 합니다.

예제를 기준으로 보면 FOR EACH ROW는 EMP 테이블에 UPDATE되는 ROW의 개수만큼 수행됩니다.

UPDATE문이 4개의 ROW를 수정한다면 TRIGGER도 4번 실행 됩니다.

FOR EACH ROW를 생략하면 STATEMENT LEVEL TRIGGER를 의미하며 해당 문장당 1번 실행 됩니다.

위의 예제는 ②를 보면 처리되는 각 ROW의 데이터에 접근하는 방식이기 때문에 STATEMENT LEVEL로 동작할 수는 없고 반드시 ROW LEVEL로 동작 해야 합니다.

<참고> 이전 차시인 FUNCTION시간에도 잠깐 설명을 드렸지만 PL/SQL BLOCK의 실행횟수에 대해서는 여러분은 민감하게 생각해야 한다.

여러분이 만든 함수가 SELECT 에 사용되어 조회되는 ROW이 개수 만큼 수행되거나 여러분이 만든 TRIGGER가 처리되는 ROW 개수 만큼 실행되는 경우 처리 성능 및 처리 시간과 관련하여 민감하게 주의를 기울여야 합니다.

배보다 배꼽이 더큰경우가 생길수 있다

② 조건문에 :NEW.SAL > 9000 는 어떤 의미일까?

TRIGGER를 학습할 때 :NEW 와 :OLD 2개를 잘 기억해두어야 합니다.

NEW 와 OLD는 이전에 학습했던 RECORD라고 생각하시면 이해가 쉽습니다.

PL/SQL 엔진이 자동으로 선언해주는 테이블을 참조하는 레코드 유형

레코드안의 필드(FIELD)를 참조할 때는 레코드명.필드명(EX R_EMP.SAL)을

사용했었죠! NEW.SAL 와 R_EMP.SAL이 유사합니다. 레코드와 구분 하기 위해 앞에 : (콜론)을 붙여 :NEW.SAL 로 사용 합니다.

연산	:OLD	:NEW
INSERT	X	O
DELETE	O	X
UPDATE	O	O

위의 표를 보면 쉽게 이해 할수 있다.

INSERT는 새로운 데이터를 삽입하므로 :NEW 만 존재

DELETE는 기존의 데이터를 삭제하므로 :OLD 만 존재

UPDATE는 기존 데이터를 변경하므로 :OLD 와 :NEW가 존재.

:NEW 와 :OLD를 가지고 변화되는 값을 제어 또는 추적할수 있다.

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

예제를 보면 신규로 변경하려는 값이 9000을 초과하면
TRIGGER에서 강제로 해당값을 9000으로 변경 제어 한다.

< 참고> 우리회사의 급여는 9000을 초과할수 없다는 것은 우리 회사의
BUSINESS RULE 입니다. BUSINESS RULE을 준수하는 방법은 3가지가 있다.

- (1) 선언적 제약사항
- (2) APPLICATION LOGIC
- (3) TRIGGER

선언적 제약사항은 PRIMARY KEY 나 CHECK, NOT NULL등의 데이터베이스상의
제약사항 입니다. 선언적 제약사항은 비교적 간단한 RULE의 구현이 가능합니다.

APPLICATION LOGIC은 개발되는 APPLICATION 프로그램 내부에 코딩으로
구현하는 형태 입니다. 선언적 제약사항에 비해 복잡한 BUSINESS RULE을
구현할수 있다.

TRIGGER는 PL/SQL로 개발되며 DBMS서버내에 저장되어 APPLICATION LOGIC
처럼 복잡한 BUSINESS RULE을 구현 한다..

비교적 간단한 BUSINESS RULE은 (1) 선언적 제약사항을 사용하는 것이 효율적
복잡한 BUSINESS RULE은 (2) APPLICATION LOGIC 나 (3) TRIGGER를 사용

BUSINESS RULE이 APPLICATION 프로그램내에 존재해야 하는 경우에는 (2)
DBMS 서버 내에 존재해야 하는 경우에는(3)을 고려

TRIGGER는 복잡한 BUSINESS RULE을 DBMS 서버내에서 관리할 때 사용



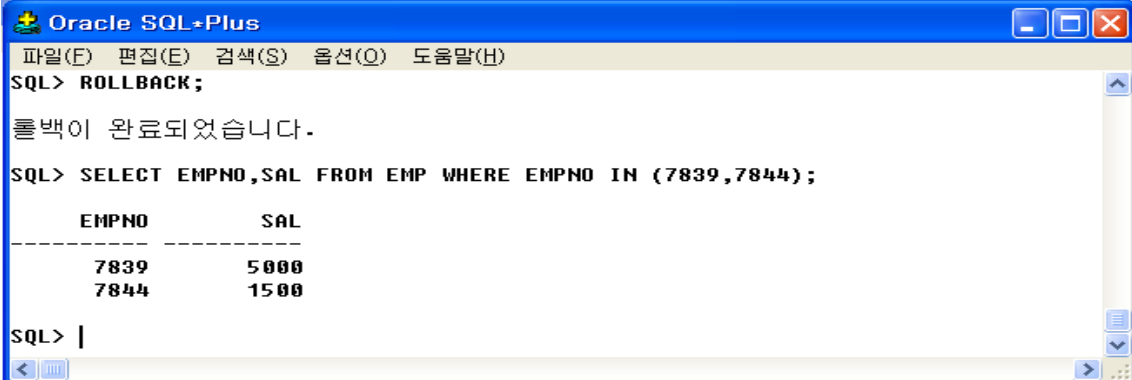
```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> @8_TRIGGER_1.SQL
트리거가 생성되었습니다.
SQL> UPDATE EMP SET SAL = 9500 WHERE EMPNO IN (7839,7844);
2 행이 갱신되었습니다.
SQL> SELECT EMPNO,SAL FROM EMP WHERE EMPNO IN (7839,7844);
   EMPNO      SAL
-----
   7839      9000
   7844      9000
SQL>
```

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

2 ROW에 대해 급여를 9500이상으로 수정하는 UPDATE를 수행한후
결과를 조회해보면 급여는 최고 상한선인 9000으로 되어있는 것을 확인.

[질문] TRIGGER에서 TRANSACTION은 어떻게 될까?

TRIGGER내에서는 TRANSACTION의 종결자(COMMIT,ROLLBACK)을 사용할수 없습니다.위의 예처럼 메인 연산이 수행되면 함께 수행되고 메인 연산이 취소되면 함께 취소 된다.



The screenshot shows the Oracle SQL*Plus interface. The command prompt shows 'SQL> ROLLBACK;' followed by the message '롤백이 완료되었습니다.' (Rollback completed). Below this, the command 'SQL> SELECT EMPNO,SAL FROM EMP WHERE EMPNO IN (7839,7844);' is entered. The result is displayed as a table with two columns: EMPNO and SAL. The data rows are 7839 with salary 5000, and 7844 with salary 1500.

EMPNO	SAL
7839	5000
7844	1500

8-2-2 TRIGGER 특징

TRIGGER는 파라미터가 없고 ORACLE에 의해서 자동으로 호출(실행)

TRIGGER의 주용도

- PREVENT INVALID TRANSACTION
- COMPLEX BUSINESS RULE SUPPORT
- COMPLEX INTEGRITY SUPPORT
- EVENT LOGGING

TRIGGERING EVENT/TIMING/LEVEL

- EVENT INSERT,DELETE,UPDATE
- TIMING BEFORE,AFTER
- LEVEL ROW-LEVEL(FOR EACH ROW),STATEMENT-LEVEL

TIGGER내에서는 TRANSACTION 제어문(COMMIT,ROLLBACK,SAVEPOINT)을 사용 할수 없는 이유는 TRIGGER의 TRASNACTON은 TRIGGERING을 유발한 DML의 TRANSACTION에 종속되기 때문

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

TRIGGER 자체도 내부적인 PROCESSING을 유발하기 때문에 빈번한 수행은 수행 속도에 영향을 줄수 있다.

8-2-3 TRIGGER 생성

[비즈니스룰] 우리회사는 신규 입사자의 직군이 CLERK 이나 SALESMAN인 경우 노조에 자동으 가입 해야 하고 퇴직시에는 퇴직자 명단에 항상 등록 된다. 기존 급여 시스템의 기능적 오류로 인해 사원의 급여 항목이 종종 마이너스 로 바뀌게 된다. 마이너스 금액으로 수정되는 경우 원래의 급여를 되돌려 놓아야 한다.

이런 BUSINESS RULE을 수행하는 TRIGGER를 만들어 보시다.

8_TRIGGER_2.SQL

퇴사자 와 노조 명단의 테이블을 생성 합니다.

```
-- -----  
-- 퇴사자 명단 TABLE  
-- -----  
  
CREATE TABLE RETIRED_EMP(  
    EMPNO          NUMBER(4) NOT NULL,      -- 사번  
    ENAME          VARCHAR2(10),            -- 이름  
    JOB            VARCHAR2(9),              -- 직군  
    RETIRED_DATE    DATE,                    -- 퇴사일  
);  
  
-- -----  
-- 노조 명단 TABLE  
-- -----  
  
CREATE TABLE LABOR_UNION(  
    EMPNO          NUMBER(4) NOT NULL,      -- 사번  
    ENAME          VARCHAR2(10),            -- 이름  
    JOB            VARCHAR2(9),              -- 직군  
    ENROLL_DATE    DATE,                    -- 퇴사일  
);
```

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

1

```
CREATE OR REPLACE TRIGGER TRG_EMP_CHANGE
BEFORE INSERT OR DELETE OR UPDATE OF SAL ON EMP
FOR EACH ROW
DECLARE -- DECLARE가 있는....
```

```
BEGIN
```

```
-- TRIGGERING EVENT를 구분한다 INSERTING,DELETING,UPDATING으로
```

```
-- 직군이 CLERK 과 SALESMAN 인경우 입사시에 자동으로 노조 명단에 등록
```

```
IF INSERTING AND :NEW.JOB IN ('CLERK','SALESMAN') THEN
```

```
INSERT INTO LABOR_UNION(EMPNO,ENAME,JOB,ENROLL_DATE) VALUES(:NEW.EMPNO,:NEW.ENAME,:NEW.JOB,SYSDATE);
```

```
ELSIF DELETING THEN -- 퇴사시 퇴직자 명단에 등록
```

```
BEGIN
```

```
INSERT INTO RETIRED_EMP(EMPNO,ENAME,JOB,RETIRED_DATE)
VALUES(:OLD.EMPNO,:OLD.ENAME,:OLD.JOB,SYSDATE);
```

```
DELETE FROM LABOR_UNION WHERE EMPNO = :OLD.EMPNO;
```

```
EXCEPTION
WHEN OTHERS THEN
NULL;
```

```
END;
```

```
ELSIF UPDATING THEN
```

```
IF :NEW.SAL < 0 THEN -- 마이너스 급여가 들어오는 경우 원래 급여로 치환
```

```
:NEW.SAL := :OLD.SAL;
```

```
END IF;
```

```
END IF;
```

```
END;
```

2

① TIMING은 BEFORE 입니다. 메인 연산보다 TRIGGER 가 먼저 실행 됩니다.

TRIGGER 이벤트는 INSERT,DELETE,UPDATE 3개입니다.

OR 로 3개의 이벤트를 1개의 TRIGGER에서 처리합니다.

<TIP : OR로 여러 TRIGGER 이벤트를 나열하는 것을 단순하지만 기억해두시길>

DECLARE ~ BEGIN은 선언부 입니다.

DECLARE 구문은 생략 가능합니다.

② IF 조건문을 보면 INSERTING, DELETING, UPDATING 키워드를 사용하여

여러 개의 TRIGGER 이벤트를 각각 처리 합니다.

INSERT 이벤트시

신규 입사자의 직군이 CLERK ,SALESMAN인 경우 노조 명단에 등록

DELETE 이벤트시

퇴직시 퇴직자 명단에 등록하고 노조 명단에서 삭제

TRIGGER 내에서도 NESTED BLOCK을 사용하고 예외처리 구문을 추가할수

있습니다.

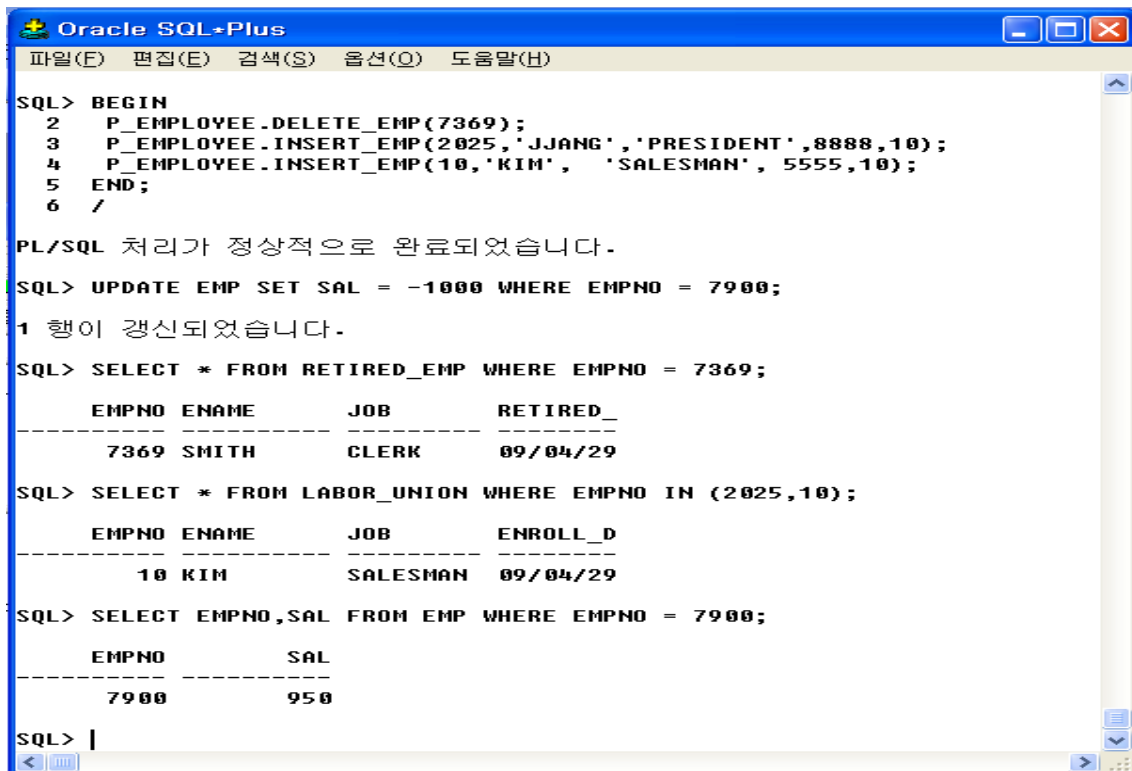
UPDATE 이벤트시

마이너스 급여로 수정시 원래의 급여로 강제로 되돌리는 구문

자 실습을 해봅시다.

<8_TRIGGER_2.SQL 발췌>

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)



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

SQL> BEGIN
2  P_EMPLOYEE.DELETE_EMP(7369);
3  P_EMPLOYEE.INSERT_EMP(2025,'JJANG','PRESIDENT',8888,10);
4  P_EMPLOYEE.INSERT_EMP(10,'KIM','SALESMAN',5555,10);
5  END;
6  /

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL> UPDATE EMP SET SAL = -1000 WHERE EMPNO = 7900;

1 행이 갱신되었습니다.

SQL> SELECT * FROM RETIRED_EMP WHERE EMPNO = 7369;

   EMPNO ENAME      JOB      RETIRED_
-----
   7369 SMITH      CLERK      09/04/29

SQL> SELECT * FROM LABOR_UNION WHERE EMPNO IN (2025,10);

   EMPNO ENAME      JOB      ENROLL_D
-----
    10 KIM        SALESMAN  09/04/29

SQL> SELECT EMPNO,SAL FROM EMP WHERE EMPNO = 7900;

   EMPNO      SAL
-----
   7900      950

SQL> |
```

TRIGGER가 생성됩니다.

이전 차시에서 생성한 PACKAGE에서 1명의 사원 데이터 삭제, 2명의 사원 신규 등록을 합니다.

UPDATE를 사용하여 마이너스 급여로 수정 합니다.

SELECT를 통해 해당 결과를 조회해보면

삭제된 7369 사원은 퇴직자 명단에 등록 되었습니다.

2명의 사원(2025,10)이 신규 등록 되었지만 10번 사원만이 노조에 가입합니다.

조건에 의해서 걸러진거죠!

7900사원의 급여를 조회해보면 마이너스가 아닌 원래의 급여로 되돌려져 있죠!

III.

1.다음 설명중 틀린 것은

- ① PACKAGE는 연관된 PROCEDURE/FUNCTION/SEQUENCE/SYNONYM 객체의 묶음이다.
- ② PACKAGE HEADER 영역은 구성요소에대한 명세(SPECIFICATION)서 입니다.
즉 구성요소에 대한 정의를 선언하는 영역으로 구체적인 소스코드를 가지고 있지는 않다.

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

자동으로 취소가 된다. 메인 연산이 수행되면 자동으로 TRIGGER도 수행되므로 TRIGGER에서 발생한 연산은 메인 연산의 TRANSACTION안에 포함되기 때문이다.

정답 ② ROW LEVEL TRIGGER는 FOR EACH ROW를 명시해야 하지만
STATEMENT LEVEL TRIGGER는 FOR EACH ROW를 생략하면
자동으로 STATEMENT LEVEL TRIGGER가 정의 됩니다.

[학습정리]

1. PACKAGE는 연관된 PROCEDURE/FUNCTION/CURSOR/VARIABLE이 묶음이다.
2. PACKAGE는 (1)PACKAGE HEADER 영역 과(2) PACKAGE BODY 영역으로 구성된다.
 - (1) PACKAGE HEADER 영역은 구성요소에대한 명세(SPECIFICATION)서입니다.
 - (2) PACKAGE BODY 영역은 HEADER 영역에서 정의한 명세에 대한 구체적인 소스 코드를 정의 합니다
3. PACKAGE HEADER영역에서 정의한 PROCEDURE/FUNCTION/VARIABLE은 PUBLIC 영역으로 PACKAGE 외부에서 직접 호출하여 사용할수 있다.
PACKAGE BODY 영역에만 정의한 PROCEDURE/FUNCTION/VARIABLE은 PRIVATE 영역으로 PACKAGE 내부에서만 사용할수 있고 외부에서 직접 호출하여 사용할수 없다.
4. 참조되는 OBJECT의 상태가 변경이 발생하게 되면 참조하는 OBJECT에도 변경 사항이 발생 합니다.
5. TRIGGER 이벤트에 따라 4가지 유형의 TRIGGER로 구분된다.
 - DATABASE TRIGGER , DDL TRIGGER
 - DML TRIGGER , INSTAED OF TRIGGER

IV.

[심화학습]

데이터베이스 프로그래밍-PL/SQL Named Block (8차시)

[토론과제]