트리거

- 이벤트처리 (~했을때 수반되는 처리)
- 특정테이블에 이벤트 (insert, delete, update)가 발생했을 시
  다른 테이블에 연관된 내용을 변경하도록 하는일.

형식)

```
CREATE [OR REPLACE] TRIGGER 트리거명
(BEFORE|AFTER) (INSERT|DELETE|UPDATE)  -- 이벤트
ON 테이블명  -- 이벤트가 발생하는 테이블
[FOR EACH ROW] -- 실행될 문장 행에 각각 적용.
[WHEN 조건식]
BEGIN
  이벤트 발생시 실행할 문장(주로 DML 문장); -- 이벤트 핸들러
END;
```

문제) 사원 테이블에 사원 정보가 새로 입력될 때 마다 입사 환영메시지를 출력하시오.

```
?
1  DROP TABLE EMP02;
2
3  CREATE TABLE EMP02
4  AS(
5      SELECT EMPNO
6           , ENAME
7           , DEPTNO
8       FROM EMP
9      WHERE1 = 0
10 );
11
12 CREATE OR REPLACE TRIGGER TR_WELCOME
13 AFTER INSERT ON EMP02
14 BEGIN
15   DBMS_OUTPUT.PUT_LINE('WELCOME KH');
16 END;
17 /
18
19 INSERT INTO EMP02 VALUES(7002, '길라임',20);
20 WELCOME KH
```

```
21
22  1 개의 행이 만들어졌습니다.
23  INSERT INTO EMP02 VALUES(7004, '김주원', 40);
24  WELCOME KH
25
26  1 개의 행이 만들어졌습니다.
```

문제) 사원 테이블에 사원 정보가(EMPNO, ENAME, SAL) 입력되었을 때 급여 테이블(NO, EMPNO, SAL)에 그 사원에 대한 급여 정보를 자동으로 입력하도록 트리거를 설계하시오.

```
?
1   CREATE TABLE EMP03
2   (
3         EMPNO NUMBER(4) PRIMARY KEY
4       , ENAME VARCHAR2(15)
5       , SAL NUMBER(7,2)
6   );
7
8   CREATE TABLE SALARY
9   (
10        NO NUMBER PRIMARY KEY
11      , EMPNO NUMBER(4)
12      , SAL NUMBER(7,2)
13  );
14
15  CREATE SEQUENCE SALARY_SEQ
16        START WITH 1
17        INCREMENT BY 1
18        NOCYCLE
19        NOCACHE;
20
21 ---> INSERT INTO EMP03 VALUES (8000, '홍길동', 3000);
22 ---> INSERT INTO SALARY VALUES (1, 8000, 3000);
23
24  INSERT INTO EMP03 VALUES(8000, '홍길동', 3000);
25
26  CREATE OR REPLACE TRIGGER TR_SALARY
27  AFTER INSERT ON EMP03
28   FOR EACH ROW
29   BEGIN
```

```
30     INSERT INTO SALARY    VALUES(SALARY_SEQ.NEXTVAL, :NEW.EMPNO, :NEW.SAL);
31    END;
32    /
33
34    SELECT * FROM EMP03;
35
36       EMPNO ENAME                    SAL
37 ---------- --------------- ----------
38        8000 홍길동                  3000
39
40    SELECT * FROM SALARY;
41
42          NO       EMPNO         SAL
43 ---------- ---------- ----------
44           1        8000        3000
```

<바인드변수> 2 가지

- FOR EACH ROW 를 사용해야 함

:NEW - 새로 입력된(INSERT, UPDATE) 데이터

:OLD - 기존 데이터

사용법) :NEW.EMPNO

문제) 테이블 만들기

1. <상품테이블>
2. <입고테이블>
3. 제품이 입고되면 상품재고에 자동으로 입력되게 함.
4. 입력, 수정, 삭제되면 재고에 변화를 줘야 함.

1. 입력 트리거 (입고 테이블에 상품이 입력되었을 때 재고수량 증가)

   예) 입고 테이블에 모니터가 10 개 입고 되었을 때 자동으로 상품의 재고가 변경.

2. 수정 트리거 (입고 테이블에 상품의 재고정보가 변경되었을 때 재고수량 변경)

   예) 두 번째 입고된 15 를 10 으로 변경하면 상품테이블의 재고수량 25 가 20 으로 변경

3. 삭제 트리거(입고 테이블에 입력된 행이 삭제되었을 때 그 수만큼 재고수량 감소)

예) 첫 번째 입고된 행(수량 10 입력)이 삭제되면 삼풍 테이블의 재고수량이 20 에서 10 으로 변경

```
?
1   CREATE TABLE 상품
2   (
3         상품코드 CHAR(4) CONSTRAINT 상품_PK PRIMARY KEY -- a001
4       , 상품명 VARCHAR2(15) NOT NULL
5       , 제조사 VARCHAR2(15)
6       , 소비자가격 NUMBER
7       , 재고수량 NUMBER DEFAULT 0
8   );
9
10  --상품등록
11  INSERT INTO 상품(상품코드, 상품명, 제조사, 소비자가격)
12  VALUES('a001', '마우스', 'LC', 1000);
13
14  INSERT INTO 상품 VALUES('a002', '키보드', '삼송', 2000, 0);
15
16  INSERT INTO 상품 VALUES('a003', '모니터', 'HB', 10000, 0);
```

```
?
1   CREATE TABLE 입고
2   (
3         입고번호 NUMBER PRIMARY KEY
4       , 상품코드 CHAR(4) REFERENCES 상품(상품코드)
5       , 입고일자 DATE
6       , 입고수량 NUMBER
7       , 입고단가 NUMBER
8       , 입고금액 NUMBER
9   );
10
11  CREATE SEQUENCE 입고_SEQ
12        START WITH 1
13        INCREMENT BY 1
14        NOCYCLE
15        NOCACHE;
16
17CREATE OR REPLACE PROCEDURE SP_PRO_INSERT(
18        CODE CHAR
```

```
19      , SU NUMBER
20      , WON NUMBER
21)
22  IS
23  BEGIN
24    INSERT INTO 입고
25      VALUES(
26            입고_SEQ.NEXTVAL
27          , CODE
28          , SYSDATE
29          , SU
30          , WON
31          , SU*WON
32      );
33  END;
34  /
```

```
?
1  -- TRIGGER INSERT
2    CREATE OR REPLACE TRIGGER TR_PRODUCT_INSERT
3    AFTER INSERT ON 입고
4    FOR EACH ROW
5    BEGIN
6       UPDATE 상품
7       SET 재고수량 = 재고수량 + :NEW.입고수량
8        WHERE 상품코드 = :NEW.상품코드;
9    END;
10  /
11  /*
12  UPDATE 입고 SET
13         입고금액 = 입고수량 * 입고단가;
14  ORA-04091: SCOTT.입고 테이블이 변경되어 트리거/함수가 볼 수
15없습니다.
16  ORA-06512: "SCOTT.TR_PRODUCT_INSERT",  6 행
17  ORA-04088: 트리거 'SCOTT.TR_PRODUCT_INSERT'의 수행시 오류
18  */
19----> 프로시저로 INSERT 함으로 해결
```

```
20
21-- TRIGGER UPDATE
22 CREATE OR REPLACE TRIGGER TR_PRODUCT_UPDATE
23 AFTER UPDATE ON 입고
24 FOR EACH ROW
25 BEGIN
26   UPDATE 상품 SET
27         재고수량 = 재고수량 - :OLD.입고수량 + :NEW.입고수량
28    WHERE 상품코드 = :OLD.상품코드;
29 END;
30 /
31 -- TRIGGER DELETE
32 CREATE OR REPLACE TRIGGER TR_PRODUCT_DELETE
33 AFTER DELETE ON 입고
34 FOR EACH ROW
35 BEGIN
36   UPDATE 상품 SET
37         재고수량 = 재고수량 - :OLD.입고수량
38    WHERE 상품코드 = :NEW.상품코드;
39 END;
40 /
41
42 EXEC SP_PRO_INSERT('a002', 20, 3000);
43
44 EXEC SP_PRO_INSERT('a002', 10, 3000);
45
46 UPDATE 입고 SET 입고수량 = 15 WHERE 입고번호 = 7;
47
48 DELETE FROM 입고 WHERE 입고번호 = 6;
49
50 SELECT * FROM 상품;
51
   SELECT * FROM 입고;
```

```

?
1 상품 상품명          제조사          소비자가격    재고수량
2 ---- --------------- --------------- ---------- ----------
```

```
3  a001 마우스          LC                  1000          0
4  a002 키보드          삼송                 2000         30
5  a003 모니터          HB                  10000         30
6
7
8   입고번호 상품 입고일자   입고수량   입고단가   입고금액
9  ---------- ---- -------- ---------- ---------- ----------
10         13 a002 15/05/15          20       3000      60000
11          7 a003 15/05/15          15       5000      50000
12         14 a002 15/05/15          10       3000      30000
13          8 a003 15/05/15          15       5000      75000
```

```
  1  create or replace function fn_salary_ename(
  2     v_ename in employee.ename%type)
  3     return number
  4   is
  5   v_salary number(7,2);
  6   begin
  7   select salary into v_Salary
  8   from employee
  9   where ename = v_ename;
 10   return v_salary;
 11*  end;
```

함수가 생성되었습니다.

```
SQL> variable v_salary number;
SQL> execute :v_salary := fn_salary_ename('SCOTT');
```

PL/SQL 처리가 정상적으로 완료되었습니다.

```
SQL> print v_salary;

  V_SALARY
----------
      3000
```

DML trigger

after trigger
table 에 insert, update, delete 등의 작업이 일어난 후에 작동
before trigger
table 이나 view 에 이벤트가 작동하기 전에 작동한다.
table, view 둥다 작동하는데 주로 뷰가 업데이트 가능하도록 사용한다
insert, update, delete 세가지 이벤트로 작동

트리거는 몸체의 내용이 몇 번이나 실행되는지에 따라서 문장레벨 트리거, 행레벨 트리거로 나뉜다

문장레벨 트리거
어떤 table 에 대해 DML 문을 실행할 때 단 한번만 트리거를 발생시키는 것
행 레벨 트리거
어떤 DML 에 대해 table 의 행이 변경된 수에 따라 트리거가 발생되는 것
즉, 10 개의 행이 바뀐다면 트리거는 10 번 실행된다.
for each row 를 사용해서 문장레벨 트리거와 행 레벨 트리거를 나눈다.
행 레벨 트리거는 실제 데이터의 값을 취하기 위해서 OLD.salary, NEW.salary 와 같은 키워드를 사용한다.

OLD 와 NEW 는 둘다 임시 테이블이다. 그러니까 insert, delete, update 문이 실행되면 oracle 에서는 DML 이 실행되면 최신화 된 테이블은 NEW 라는 테이블에 저장되고, DML 이 실행되기 이전의 테이블은 잠시동안 OLD 라는 테이블에 임시로 저장되어진다.

```
//////////////////////////insert trigger//////////////////////////
create or replace trigger trigger_sampel1
        after insert
        on dept_original
        for each row
begin
        if inserting then
        dbms_output.put_line('insert trigger 발생');
        insert into dept_copy
        values(:old.dno, :old.dname, :old.loc);
        end if;
end;
//////////////////////////insert trigger//////////////////////////

//////////////////////////delete trigger//////////////////////////
create or replace trigger trigger_sampel1
        delete insert
```

```
        on dept_original
        for each row
begin
        if inserting then
        dbms_output.put_line('delete trigger 발생');
        delete from dept_copy
        where dept_copy.dno = :old.dno;
        end if;
end;
//////////////////////////////delete trigger//////////////////////////////
```