

Creating Database Triggers

1. 트리거 개요

- ☐ 트리거는 특정 이벤트가 발생할 때 마다 암시적으로 실행되는 PL/SQL 블록
 - INSERT, UPDATE, DELETE 문 등의 트리거 문이 관련 테이블에 대해 실행될 때 암시적으로 실행
- ☐ 트리거는 데이터베이스 트리거 또는 응용 프로그램 트리거로 구분

2. 트리거 설계 지침

- ☐ 특정 작업을 수행할 때 관련 작업도 수행하려면 트리거 사용
- ☐ 명령문을 실행하는 사용자 또는 응용 프로그램에 관계없이 트리거 문에서 실행할 중앙 집중식 전역 작업에 대해서만 데이터베이스 트리거 사용
- ☐ 이미 오라클 데이터베이스에 내장된 함수를 복제하거나 바꾸는 트리거는 정의하지 말아야 함
- ☐ 트리거를 과다하게 사용하면 상호 종속성이 복잡해져 대형 응용 프로그램에서 트리거 유지 관리 어려움
 - 필요할 때만 트리거를 사용하고 순환 및 계단식 효과를 주의

3. 데이터베이스 트리거

3-1. 트리거 작성

구성 요소	설 명	가능한 값
트리거 타이밍	트리거 이벤트에 따라 트리거를 실행하는 시기	BEFORE AFTER INSTEAD OF
트리거 이벤트	트리거를 실행하는 테이블 또는 뷰의 데이터 조작 작업	INSERT UPDATE DELETE
트리거 유형	트리거 본문 실행 횟수	문 (Statement) 행 (Row)
트리거 본문	트리거가 수행하는 작업	모든 PL/SQL 블록

3-2. 트리거 구성 요소 : 트리거 타이밍

☐ BEFORE 트리거 사용

- 트리거 작업으로 트리거 문의 완료 허용 여부를 결정해야 하는 경우
- INSERT 또는 UPDATE 트리거 문을 완료하기 전에 열 값을 구해야 하는 경우

☐ AFTER 트리거 사용

- 트리거 작업을 실행하기 전에 트리거 문을 완료할 경우
- 이미 BEFORE 트리거가 있고 동일한 트리거 문에서 AFTER 트리거가 다른 작업을 수행할 수 있을 경우

☐ INSTEAD OF 트리거 사용

- 수정 불가능 해서 SQL DML 문을 통해 직접 수정할 수 없는 뷰는 이 트리거를 사용하여 그대로 수정 가능

3-3. 트리거 구성 요소 : 트리거 이벤트

- ☐ 트리거 이벤트 또는 트리거 문은 테이블의 INSERT, UPDATE, DELETE 문
 - 트리거 이벤트가 UPDATE면 열 목록을 포함시켜 트리거 실행을 위해 변경해야 할 열을 식별할 수 있음
 - INSERT, DELETE 문은 전체 행에 영향을 주기 때문에 이들에 대한 열 목록은 지정할 수 없음

```
..... UPDATE OF sal ....
```

- 트리거 이벤트는 여러 DML 작업 포함 가능

```
..... INSERT or UPDATE or DELETE
```

```
..... INSERT or UPDATE OF job ....
```

3-4. 트리거 구성 요소 : 트리거 유형

- ☐ 트리거 문의 영향을 받는 모든 행에 대해 한 번 실행하거나 영향을 받는 행 수와 관계없이 트리거 문에 대해 한 번 실행하도록 트리거 작업의 실행 횟수 지정 가능
- ☐ 문장 트리거
 - 영향을 받는 행이 없더라도 트리거 이벤트 대신 문장 트리거를 한 번 실행
 - 영향을 받는 행의 데이터 또는 트리거 이벤트 자체에서 제공하는 데이터에 트리거 작업이 종속되지 않을 경우에는 문장 트리거가 유용
- ☐ 행 트리거
 - 테이블이 트리거 이벤트의 영향을 받을 때마다 행 트리거 실행
 - 트리거 이벤트가 행에 영향을 주지 않으면 행 트리거를 실행하지 않음
 - 영향을 받는 행의 데이터 또는 트리거 이벤트 자체에서 제공하는 데이터에 트리거 작업이 종속될 경우에는 행 트리거가 유용

3-5. 트리거 구성 요소 : 트리거 본문

- ☐ 트리거가 수행하는 작업
- ☐ 트리거 작업은 트리거 이벤트가 실행될 때 수행해야 할 작업을 정의

3-6. 실행 순서

- ☐ 단일 행이 조작될 때의 테이블에 대한 트리거 실행

```
INSERT INTO dept
VALUES (50, 'EDUCATION', 'NEW YORK');
```

DEPTNO	DNAME	DNAME
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	EDUCATION	NEW YORK

BEFORE 문장 트리거

BEFORE 행 트리거

AFTER 행 트리거

AFTER 문장 트리거

3-6. 실행 순서

- ☐ 여러 행이 조작될 때의 테이블에 대한 트리거 실행

```
UPDATE emp
SET      sal = sal * 1.1
WHERE    deptno = 30;
```

EMPNO	ENAME	DEPTNO	
7839	KING	30	BEFORE 문장 트리거
7698	BLAKE	30	BEFORE 행 트리거
			AFTER 행 트리거
			BEFORE 행 트리거
			AFTER 행 트리거
7788	EDUCATION	30	BEFORE 행 트리거
			AFTER 행 트리거
			AFTER 문장 트리거

4. 문장 트리거

4-1. 문장 트리거 작성 구문

```
CREATE [OR REPLACE] TRIGGER trigger_name
    timing
    event1 [OR event2 OR event3]
    ON table_name
    trigger_body
```

4-2. 문장 트리거 작성

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON emp
BEGIN
    IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')
        OR (TO_CHAR(SYSDATE, 'HH24') NOT BETWEEN '08' AND '18'))
    THEN RAISE_APPLICATION_ERROR (-20500,
        'You may only insert into EMP during normal hours. ');
    END IF;
END;
/
```

```
INSERT INTO emp (empno, ename, deptno)
VALUES (7777, 'BAUSENS', 40);
```

4-3. 트리거 이벤트 결합

- ☐ 트리거 본문 안에서 특수한 조건문 술어 **INSERTING**, **UPDATING**, **DELETING**을 이용하여 여러 트리거 이벤트를 하나로 결합

```
CREATE OR REPLACE TRIGGER secure_emp
AFTER INSERT OR UPDATE OR DELETE ON emp
BEGIN
    IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN'))
    OR (TO_CHAR(SYSDATE, 'HH24') NOT BETWEEN '08' AND '18')
    THEN
        IF DELETING THEN
            RAISE_APPLICATION_ERROR (-20502,
            'You may only delete from EMP during normal hours.');
```

```
        ELIF INSERTING THEN
            RAISE_APPLICATION_ERROR (-20500,
            'You may only insert into EMP during normal hours.');
```

```
        ELIF UPDATING('SAL') THEN
            RAISE_APPLICATION_ERROR (-20503,
            'You may only update SAL during normal hours.');
```

```
        ELSE
            RAISE_APPLICATION_ERROR (-20504,
            'You may only update EMP during normal hours.');
```

```
    END IF;
END IF;
END;
/
```

5. 행 트리거

5-1. 행 트리거 작성 구문

```
CREATE [OR REPLACE] TRIGGER trigger_name
    timing
    event1 [OR event2 OR event3]
    ON table_name
    [REFERENCING OLD AS old | NEW AS new]
    FOR EACH ROW
    [WHEN condition]
    trigger_body
```

5-1. 행 트리거 작성 구문 (계속)

<i>trigger_name</i>	트리거 이름
<i>timing</i>	트리거 이벤트에 따라 트리거를 실행하는 시기 - BEFORE - AFTER
<i>event</i>	트리거를 실행하는 데이터 조작 작업 - INSERT - UPDATE [OF <i>column</i>] - DELETE
<i>table_name</i>	트리거와 관련된 테이블
REFERENCING	현재 행의 기존 값과 새로운 값의 상관 이름 지정 (기본값은 OLD와 NEW)
FOR EACH ROW	트리거를 행 트리거로 지정
WHEN	트리거 제한 사항 지정 (각 행을 평가하여 트리거 본문의 실행 여부 결정)
<i>trigger body</i>	- DECLARE 또는 BEGIN으로 시작하여 END로 끝남 - 트리거에 의해 수행되는 작업 또는 프로시저 호출을 정의

5-2. 행 트리거 작성

```
CREATE OR REPLACE TRIGGER derive_commission_pct
  BEFORE INSERT OR UPDATE OF sal ON emp
  FOR EACH ROW
BEGIN
  IF NOT (:NEW.JOB IN ('MANAGER' , 'PRESIDENT'))
    AND :NEW.SAL > 5000
  THEN
    RAISE_APPLICATION_ERROR
      (-20202, 'EMPLOYEE CANNOT EARN THIS AMOUNT') ;
  END IF;
END ;
/
```

```
UPDATE emp
SET      sal = 6500
WHERE    ename = 'MILLER' ;
```

5-3. OLD 및 NEW 수식자 사용

```
CREATE TABLE audit_emp_table (  
  user_name      VARCHAR2(10) ,  
  timestamp      DATE ,  
  id             NUMBER(5) ,  
  old_last_name  VARCHAR2(8) ,  
  new_last_name  VARCHAR2(8) ,  
  old_title      VARCHAR2(10) ,  
  new_title      VARCHAR2(10) ,  
  old_salary     NUMBER(5) ,  
  new_salary     NUMBER(5) ) ;
```

5-3. OLD 및 NEW 수식자 사용 (계속)

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
    INSERT INTO audit_emp_table
    VALUES (USER, SYSDATE, :OLD.empno, :OLD.ename, :NEW.ename,
            :OLD.job, :NEW.job, :OLD.sal, :NEW.sal);
END;
/
```

```
INSERT INTO emp ...
DELETE emp ...
UPDATE emp ...
```

```
SELECT *
FROM    audit_emp_table;
```

5-4. 행 트리거 제한

- ☐ 트리거 작업을 특정 조건에 맞는 행으로 제한하려면 **WHEN** 절 사용
- ☐ **WHEN** 절에서는 **NEW** 수식자 앞에 콜론(:) 접두어를 붙이지 않아도 됨

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF sal ON emp
FOR EACH ROW
WHEN (NEW.job = 'SALESMAN')
BEGIN
    IF INSERTING THEN
        :NEW.comm := 0;
    ELSIF :OLD.comm IS NULL THEN
        :NEW.comm := 0;
    ELSE
        :NEW.comm := :OLD.comm * (:NEW.sal/:OLD.sal);
    END IF;
END;
/
```

6. INSTEAD OF 트리거

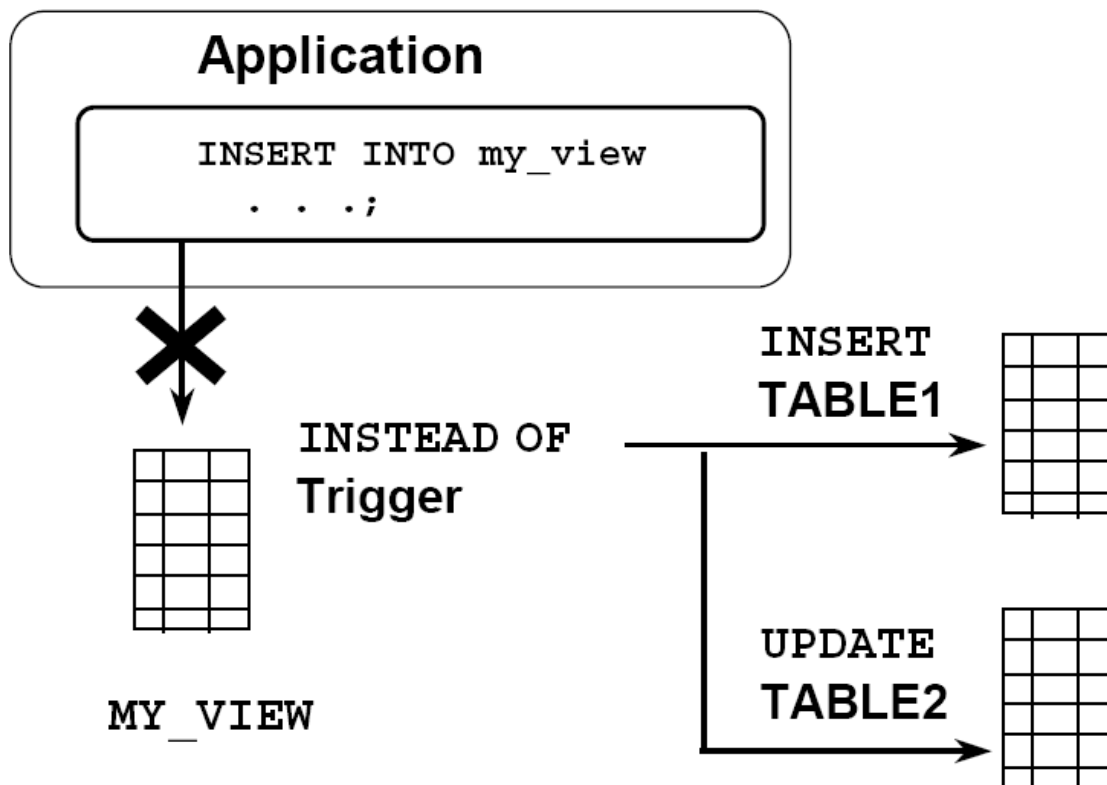
6-1. INSTEAD OF 트리거 작성 구문

```
CREATE [OR REPLACE] TRIGGER trigger_name
  INSTEAD OF
    event1 [OR event2 OR event3]
    ON view_name
    [REFERENCING OLD AS old | NEW AS new]
    [FOR EACH ROW]
  trigger_body
```

6-1. INSTEAD OF 트리거 작성 구문 (계속)

<i>trigger_name</i>	트리거 이름
INSTEAD OF	트리거가 뷰에 사용됨을 나타냄
<i>event</i>	트리거를 실행하는 데이터 조작 작업 <ul style="list-style-type: none"> - INSERT - UPDATE [OF <i>column</i>] - DELETE
<i>view_name</i>	트리거와 관련된 뷰
REFERENCING	현재 행의 기존 값과 새로운 값의 상관 이름 지정 (기본값은 OLD와 NEW)
FOR EACH ROW	<ul style="list-style-type: none"> - 트리거를 행 트리거로 지정 - INSTEAD OF 트리거는 행 트리거만 가능하므로 선택적
<i>trigger body</i>	<ul style="list-style-type: none"> - DECLARE 또는 BEGIN으로 시작하여 END로 끝남 - 트리거에 의해 수행되는 작업 또는 프로시저 호출을 정의

6-1. INSTEAD OF 트리거 작성 구문 (계속)



6-2. INSTEAD OF 트리거 작성

```
CREATE TABLE new_emps
AS SELECT empno, ename, job, sal, hiredate, deptno
FROM emp;
```

```
CREATE TABLE new_depts
AS SELECT d.deptno, d.dname, d.loc,
SUM(e.sal) tot_dept_sal
FROM emp e, dept d
WHERE e.deptno = d.deptno
GROUP BY d.deptno, d.dname, d.loc;
```

```
CREATE VIEW emp_details
AS SELECT e.empno, e.ename, e.job, e.sal, d.deptno, d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```


6-2. INSTEAD OF 트리거 작성 (작성)

```
CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emps
        VALUES (:NEW.empno, :NEW.ename, :NEW.job, :NEW.sal,
                SYSDATE, :NEW.deptno);
        UPDATE new_depts
        SET     tot_dept_sal = tot_dept_sal + :NEW.sal
        WHERE  deptno = :NEW.deptno;
```

...

6-2. INSTEAD OF 트리거 작성 (작성)

```

ELSIF DELETING THEN
    DELETE new_emps
    WHERE empno = :OLD.empno;
    UPDATE new_depts
    SET tot_dept_sal = tot_dept_sal - :OLD.sal
    WHERE deptno = :OLD.deptno;

ELSIF UPDATING ('sal') THEN
    UPDATE new_emps
    SET sal = :NEW.sal
    WHERE empno = :OLD.empno;
    UPDATE new_depts
    SET tot_dept_sal = tot_dept_sal + (:NEW.sal-:OLD.sal)
    WHERE deptno = :OLD.deptno;

...

```

6-2. INSTEAD OF 트리거 작성 (작성)

```
ELSIF UPDATING ('deptno') THEN
    UPDATE new_emps
    SET      deptno = :NEW.deptno
    WHERE    empno = :OLD.empno;

    UPDATE new_depts
    SET      tot_dept_sal = tot_dept_sal - :OLD.sal
    WHERE    deptno = :OLD.deptno;

    UPDATE new_depts
    SET      tot_dept_sal = tot_dept_sal + :NEW.sal
    WHERE    deptno = :OLD.deptno;
END IF;
END;
...
```

6-2. INSTEAD OF 트리거 작성 (작성)

```
INSERT INTO emp_details (empno, ename, sal, deptno)
VALUES (9001, 'ABBOTT', 3000, 10);
```

7. 트리거 관리

7-1. 트리거 활성화 및 비활성화

- ☐ 트리거 비활성화 또는 활성화

```
ALTER TRIGGER trigger_name DISABLE | ENABLE
```

- ☐ 테이블에 대한 모든 트리거 비활성화 또는 활성화

```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS
```

- ☐ 트리거를 처음 작성하면 자동으로 활성화
- ☐ ALTER TRIGGER 구문을 사용하여 특정 트리거 비활성화
- ☐ ALTER TABLE 구문을 사용하여 테이블의 모든 트리거 비활성화

7-2. 트리거 컴파일

```
ALTER TRIGGER trigger_name COMPILE
```

- ☐ ALTER TRIGGER 명령을 사용하여 유효하지 않은 트리거를 명시적으로 재컴파일
 - 이 경우 트리거의 유효성 여부와 관계없이 트리거가 재컴파일

7-3. 트리거 코드 표시

USER_TRIGGERS

이름	설 명
TRIGGER_NAME	트리거 이름
TRIGGER_TYPE	유형은 BEFORE, AFTER, INSTEAD OF
TRIGGERING_EVENT	트리거를 실행하는 DML 작업
TABLE_NAME	데이터베이스 테이블 이름
REFERENCING_NAMES	:OLD, :NEW에 사용되는 이름
WHEN_CLAUSE	사용된 WHEN 절
STATUS	트리거 상태
TRIGGER_BODY	수행하는 작업

7-3. 트리거 코드 표시 (계속)

```
SELECT trigger_name, trigger_type, triggering_event,  
       table_name, referencing_names, when_clause, status,  
       trigger_body  
FROM   user_triggers  
WHERE  trigger_name = 'DERIVE_COMMISSION_PCT';
```


7-4. 트리거 삭제

```
DROP TRIGGER trigger_name
```

```
DROP TRIGGER secure_emp;
```