ORACLE TRIGGEREK

A trigger egy aktualizálási művelet esetén végrehajtandó programrészletet definiál.
Alakja:

CREATE [OR REPLACE] TRIGGER név
{ BEFORE | AFTER | INSTEAD OF }
{ DELETE | INSERT | UPDATE [OF oszlopok] }
[OR { DELETE | INSERT | UPDATE [OF oszlopok] }]
ON tábla
[REFERENCING [OLD AS régi] [NEW AS új]]
[FOR EACH ROW]
[WHEN (feltétel)]
PL/SQL blokk;

Jelölés: a fenti szintaxis leírásban { x | y } azt jelenti, hogy x és y egyike választható.

név: a trigger neve.

BEFORE, AFTER, INSTEAD OF: az aktualizálási művelet előtt, után, vagy helyette lép működésbe a trigger.

DELETE, INSERT, UPDATE OF: az aktualizálási művelet neve.

ON tábla: ezen tábla aktualizálásakor lép működésbe a trigger.

REFERENCING: lehetővé teszi, hogy a tábla aktuális sorának aktualizálás előtti és utáni állapotára névvel hivatkozzunk.

A módosítás előtti (OLD) és utáni (NEW) állapot elnevezésére szolgál. Szabályok:

- Csak sor szintű triggernél használható.
- OLD és NEW az alapértelmezett nevek.
  REFERENCING csak akkor kell, ha más nevet akarunk.
- A WHEN részben OLD és NEW,
  a PL/SQL blokkban :OLD és :NEW használandó.

Használati szabályok:

- INSERT esetén csak NEW,
- UPDATE esetén OLD és NEW,
- DELETE esetén csak OLD használható.
- Módosítani csak :NEW értékét lehet és csak BEFORE trigger esetén.

FOR EACH ROW: ha megadjuk, akkor a trigger a tábla minden egyes sorára lefut, amelyet az aktualizálási művelet érint (sor szintű trigger). Ha nem adjuk meg, akkor egy aktualizálási művelet esetén csak egyszer fut le a trigger (utasítás szintű trigger).

WHEN feltétel: a megadott feltétel teljesülése esetén hajtódik végre a trigger.

programblokk: egy vagy több SQL utasításból álló, vagy valamely programozási nyelven írt blokk.

KORLÁTOZÁSOK

BEFORE és AFTER triggerek nem használhatók nézettáblára.
INSTEAD OF trigger csak nézettáblára használható.

TRIGGER FORDÍTÁSA

A CREATE TRIGGER hatására a fordítás megtörténik.
Fordítási hibák megjelenítése SQL*Plus-ban:
SHOW ERRORS paranccsal.

ENGEDÉLYEZÉS, LETILTÁS

```
ALTER TRIGGER triggernév ENABLE;
ALTER TRIGGER triggernév DISABLE;
```

PÉLDÁK
/ELŐADÁS ANYAGBÓL/

1/a. példa: utasítás szintű trigger

Egyelemű segédtábla:

```
CREATE TABLE szamlalo (ertek NUMBER);
INSERT INTO szamlalo VALUES (-1);
```

Több új dolgozó felvételének esetére két triggert definiálunk. Az első nullázza a számlálót:

```
CREATE TRIGGER dolg_kezdo
  BEFORE INSERT ON dolgozo
  BEGIN
    UPDATE szamlalo SET ertek=0;
  END;
/
```

1/b. példa: sor szintű trigger

A második trigger megszámolja a 100 000-nél kisebb fizetésű új dolgozókat:

```
CREATE TRIGGER dolg_szamlal
  AFTER INSERT ON dolgozo
  FOR EACH ROW
  WHEN (NEW.fizetes < 100000)
  BEGIN
    UPDATE szamlalo SET ertek=ertek+1;
  END;
/
```

2/a. példa: fizetés módosítása

Az alábbi trigger figyelmeztet, ha egy dolgozó fizetését csökkentettük:

```
CREATE TRIGGER fizetesNemCsokken
AFTER UPDATE OF fizetes ON dolgozo
FOR EACH ROW
WHEN (OLD.fizetes > NEW.fizetes)
BEGIN
  DBMS_OUTPUT.PUT_LINE (:NEW.nev || ' fizetése csökkent!');
END;
/
```

3. példa: „naplózó” trigger

Dolgozó(adószám, név, lakcím, fizetés)
fizetésNapló(dátum, adoszám, régifiz, újfiz)

```
CREATE TRIGGER fiz_naplo
  AFTER UPDATE ON dolgozo
  FOR EACH ROW
  BEGIN
    INSERT INTO fizetesNaplo
      VALUES (SYSDATE, :OLD.adoszam,  :OLD.fizetes,
:NEW.fizetes);
  END;
/
```

PÉLDÁK
/GYAKORLATI PÉLDÁK EMP/DEPT TÁBLÁKON/

1. példa (BEFORE TRIGGER)

```
CREATE OR REPLACE TRIGGER emp_alert_trig
    BEFORE INSERT ON emp
BEGIN
    DBMS_OUTPUT.PUT_LINE('New employees are about to be
added');
END;
```

Szúrjunk be egy sort!

```
INSERT INTO emp(empno, ename, deptno) VALUES(8000, ’valaki’,
40);
```

2. példa (AFTER TRIGGER)

Hozzuk létre az új táblát, ahol a módosításokat fogjuk letárolni!

```
CREATE TABLE empauditlog (
    audit_date      DATE,
    audit_user      VARCHAR2(20),
    audit_desc      VARCHAR2(20)
);
```

Hozzuk létre a triggert!

```
CREATE OR REPLACE TRIGGER emp_audit_trig
    AFTER INSERT OR UPDATE OR DELETE ON emp
DECLARE
    v_action        VARCHAR2(20);
BEGIN
    IF INSERTING THEN
        v_action := 'Added employee(s)';
    ELSIF UPDATING THEN
        v_action := 'Updated employee(s)';
    ELSIF DELETING THEN
        v_action := 'Deleted employee(s)';
    END IF;
    INSERT INTO empauditlog VALUES (SYSDATE, USER, v_action);
END;
```

Hajtsunk végre pár műveletet! Pl.:

```
INSERT INTO emp VALUES
(9001,'SMITH','ANALYST',7782,SYSDATE,NULL,NULL,10);
```

```
INSERT INTO emp VALUES
(9002,'JONES','CLERK',7782,SYSDATE,NULL,NULL,10);

UPDATE emp SET sal = 4000.00, comm = 1200.00 WHERE empno IN
(9001, 9002);

DELETE FROM emp WHERE empno IN (9001, 9002);
```

Kérjük le az empauditlog tábla tartalmát!

```
SELECT * FROM empauditlog;
```

3. Példa (Sor szintű BEFORE TRIGGER)

```
CREATE OR REPLACE TRIGGER emp_comm_trig
    BEFORE INSERT ON emp
    FOR EACH ROW
BEGIN
    IF :NEW.deptno = 30 THEN
        :NEW.comm := :NEW.sal * 0.4;
    END IF;
END;
```

Szúrjunk be új sorokat!

```
INSERT INTO emp VALUES
(9005,'ROBERS','SALESMAN',7782,SYSDATE,3000.00,NULL,30);

INSERT INTO emp VALUES
(9006,'ALLEN','SALESMAN',7782,SYSDATE,4500.00,NULL,30);
```

Nézzük meg a beszúrt sorok tartalmát!

```
SELECT * FROM emp WHERE empno IN (9005, 9006);
```

4. Példa (Sor szintű AFTER TRIGGER)

Hozzuk létre az empchglog segédtáblát, ahol tároljuk majd a változtatásokat!

```
CREATE TABLE empchglog (
    chg_date        DATE,
    chg_desc        VARCHAR2(50)
);
```

Hozzuk létre a triggert!

```
CREATE OR REPLACE TRIGGER emp_chg_trig
    AFTER INSERT OR UPDATE OR DELETE ON emp
    FOR EACH ROW
DECLARE
    v_empno         emp.empno%TYPE;
    v_action        VARCHAR2(7);
    v_chgdesc       VARCHAR2(30);
BEGIN
    IF INSERTING THEN
        v_action := 'Added';
        v_empno := :NEW.empno;
        v_chgdesc := '';
    ELSIF UPDATING THEN
        v_action := 'Updated';
        v_empno := :NEW.empno;
        v_chgdesc := ' Updated ';

        IF NVL(:OLD.ename, '-null-') != NVL(:NEW.ename, '-null-') THEN
            v_chgdesc := v_chgdesc || 'name';
        END IF;
        IF NVL(:OLD.job, '-null-') != NVL(:NEW.job, '-null-') THEN
            v_chgdesc := v_chgdesc || 'job';
        END IF;
        IF NVL(:OLD.sal, -1) != NVL(:NEW.sal, -1) THEN
            v_chgdesc := v_chgdesc || 'salary';
        END IF;
        IF NVL(:OLD.comm, -1) != NVL(:NEW.comm, -1) THEN
            v_chgdesc := v_chgdesc || 'commission';
        END IF;
        IF NVL(:OLD.deptno, -1) != NVL(:NEW.deptno, -1) THEN
            v_chgdesc := v_chgdesc || 'department';
        END IF;
    ELSIF DELETING THEN
        v_action := 'Deleted';
        v_empno := :OLD.empno;
        v_chgdesc := '';

    END IF;
    INSERT INTO empchglog VALUES (SYSDATE,
        v_action || ' employee # ' || v_empno || v_chgdesc);
END;
```

Hajtsunk végre pár műveletet! Pl.:

```
INSERT INTO emp VALUES
(9003,'PETERS','ANALYST',7782,SYSDATE,5000.00,NULL,40);

INSERT INTO emp VALUES
(9004,'AIKENS','ANALYST',7782,SYSDATE,4500.00,NULL,40);

UPDATE emp SET comm = sal * 1.1 WHERE empno IN (9003, 9004);
```

Nézzük meg az empchglog tábla tartalmát!

```
SELECT * FROM empchglog;
```