

# Trighere

- CREATE OR REPLACE TRIGGER Print\_salary\_changes BEFORE DELETE OR INSERT OR UPDATE ON Emp\_tab FOR EACH ROW WHEN (new.Empno > 0)

```
DECLARE sal_diff number;
```

```
BEGIN
```

```
    sal_diff := :new.sal - :old.sal;
```

```
    dbms_output.put('Old salary: ' || :old.sal);
```

```
    dbms_output.put(' New salary: ' || :new.sal);
```

```
    dbms_output.put_line(' Difference ' || sal_diff);
```

```
END;
```

```
/
```

```
UPDATE Emp_tab SET sal = sal + 500.00 WHERE deptno = 10;
```

Un trigher este :bloc PL/SQL, sau procedura PL/SQL sau C sau Java, asociat cu o tabela, view, schema sau baza de date.

# Trighere

- Trigherele pot fi lansate de una din urmatoarele:
    - Instructiuni DML (DELETE, INSERT, UPDATE)
    - Instructiuni DDL (CREATE, ALTER, DROP)
    - Operatii asupra bazelor de date (SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN)
  - Un trigger se lanseaza pe baza unei instructiuni declansatoare, ce specifica:
    - Instructiunea SQL sau evenimentul sistem, sau evenimentul de baza de date, sau evenimentul DDL ce lanseaza corpul triggerului.
    - Optiunile instructiunii includ DELETE, INSERT, sau UPDATE. - Tabela, view, baza de date sau schema asociata cu triggerul.
    - O singura tabela poate fi specificata in trigger.
    - Daca se foloseste un view, atunci singura optiune este INSTEAD OF si invers folosirea lui INSTEAD OF implica un view.
- ```
DELETE FROM Emp_tab; INSERT INTO Emp_tab VALUES ( ... );  
INSERT INTO Emp_tab SELECT ... FROM ... ; UPDATE Emp_tab  
SET ... ;
```

# Trighere

- O instructiune UPDATE poate include o lista de coloane. Trigherul se lanseaza numai cind coloana respectiva este actualizata. Daca triherul omite lista de coloane, atunci el este lansat cind se actualizeaza una din coloanele tabeli respective. Pentru instructiunile declansatoare INSERT sau DELETE, nu trebuie specificata lista de coloane.
- ...BEFORE DELETE OR INSERT OR UPDATE OF ename ON Emp\_tab...
- Nu putem specifica o lista de coloane pentru UPDATE cu trgherii INSTEAD OF.
- Daca coloana specificata in UPDATE OF este o coloana obiect, atunci trigherul este declansat daca oricare din attributele obiectului este modificat.
- Nu putem specifica clauze UPDATE OF pe coloane de tip colectie.
- Folosim BEFORE pentru a modifica linia inainte ca data respectiva sa fie scrisa pe disc.
- Folosim AFTER sa obtinem si sa realizam operatii, care utilizeaza ID de linie.

# Trighere

- Nu este cunoscuta ordinea in care liniile sunt tratate de o comanda SQL, deci nu proiectam un trigger care sa depinda de ordinea liniilor din tabela respectiva.
- Cind o instructiune din corpul unui trigger implica lansarea unui alt trigger, atunci triggerii se numesc in cascada.
- Un cursor sistem este deschis pentru fiecare executie a unui trigger.
- Oracle executa toti triggerii de acelasi tip inainte de a executa triggerii de un tip diferit. Daca exista triggeri multipli de acelasi fel pe o aceeasi tabela, atunci ordinea de executie a lor este arbitrara.
- Daca dorim ca actiuni multiple sa apara intr-o anumita ordine, atunci trebuie sa dam aceste actiuni intr-un singur trigger(de ex. Triggerul apeleaza o serie de proceduri).

# Trighere

- Un view nu poate fi modificat de instructiunile UPDATE, INSERT, sau DELETE daca interogarea view query contine una din urmatoarele constructii:

- Un operator de multime
- Un operator DISTINCT
- O functie de agregare sau analitica
- O clauza GROUP BY, ORDER BY, MODEL, CONNECT BY, sau START WITH
- O expresie colectie intr-o lista din SELECT
- O subinterogare intr-o lista SELECT
- O subinterogare cu WITH READ ONLY
- Joinuri, cu unele exceptii

# Trighere

## SQL Statement

```
UPDATE t1 SET ...;
```

Fires the  
UPDATE\_T1  
Trigger

## UPDATE\_T1 Trigger

```
BEFORE UPDATE ON t1  
FOR EACH ROW  
BEGIN  
  .  
  .  
  INSERT INTO t2 VALUES (...);  
  .  
  .  
END;
```

Fires the  
INSERT\_T2  
Trigger

## INSERT\_T2 Trigger

```
BEFORE INSERT ON t2  
FOR EACH ROW  
BEGIN  
  .  
  .  
  INSERT INTO ... VALUES (...);  
  .  
  .  
END;
```

etc.

# Trighere

- CREATE TABLE Project\_tab ( Prj\_level NUMBER, Projno NUMBER, Resp\_dept NUMBER)

/

```
CREATE TABLE Emp_tab ( Empno NUMBER NOT NULL, Ename
VARCHAR2(10), Job VARCHAR2(9), Mgr NUMBER(4), Hiredate
DATE, Sal NUMBER(7,2), Comm NUMBER(7,2), Deptno
NUMBER(2) NOT NULL)
```

/

```
CREATE TABLE Dept_tab ( Deptno NUMBER(2) NOT NULL,
Dname VARCHAR2(14), Loc VARCHAR2(13), Mgr_no NUMBER,
Dept_type NUMBER)
```

/

```
CREATE OR REPLACE VIEW manager_info AS SELECT e.ename,
e.empno, d.dept_type, d.deptno, p.prj_level, p.projno FROM Emp_tab
e, Dept_tab d, Project_tab p WHERE e.empno = d.mgr_no AND
d.deptno = p.resp_dept
```

/

# Trighere

- CREATE OR REPLACE TRIGGER manager\_info\_insert  
INSTEAD OF INSERT ON manager\_info REFERENCING NEW  
AS n -- new manager information  
FOR EACH ROW DECLARE rowcnt number;  
BEGIN  
    SELECT COUNT(\*) INTO rowcnt FROM Emp\_tab WHERE  
empno = :n.empno;  
    IF rowcnt = 0 THEN INSERT INTO Emp\_tab (empno,ename)  
VALUES (:n.empno, :n.ename);  
    ELSE UPDATE Emp\_tab SET Emp\_tab.ename = :n.ename WHERE  
Emp\_tab.empno = :n.empno;  
    END IF;  
    SELECT COUNT(\*) INTO rowcnt FROM Dept\_tab WHERE  
deptno = :n.deptno; IF rowcnt = 0 THEN INSERT INTO Dept\_tab  
(deptno, dept\_type) VALUES(:n.deptno, :n.dept\_type);  
    ELSE UPDATE Dept\_tab SET Dept\_tab.dept\_type = :n.dept\_type  
WHERE Dept\_tab.deptno = :n.deptno;

# Trighere

```
END IF;
    SELECT COUNT(*) INTO rowcnt FROM Project_tab WHERE
        Project_tab.projno = :n.projno;
    IF rowcnt = 0 THEN INSERT INTO Project_tab (projno, prj_level)
        VALUES(:n.projno, :n.prj_level);
    ELSE
        UPDATE Project_tab SET Project_tab.prj_level = :n.prj_level
            WHERE Project_tab.projno = :n.projno;
    END IF;
END;
```

- Trighere pe tabelle nested:

```
CREATE OR REPLACE VIEW Dept_view AS SELECT d.Deptno,
    d.Dept_type, d.Dept_name,
    CAST (MULTISET ( SELECT e.Empno, e.Empname, e.Salary)
        FROM Emp_tab e WHERE e.Deptno = d.Deptno) AS Emplist
FROM Dept_tab d;
```

# Trighere

- CREATE OR REPLACE TRIGGER Dept\_emplist\_tr  
INSTEAD OF INSERT ON NESTED TABLE Emplist OF  
Dept\_view  
REFERENCING NEW AS Employee PARENT AS Department  
FOR EACH ROW  
*BEGIN -- The insert on the nested table is translated to an insert on  
the base table:*  
INSERT INTO Emp\_tab VALUES ( :Employee.Empno,  
:Employee.Empname, :Employee.Salary, :Department.Deptno);  
END;  
  
INSERT INTO TABLE (SELECT d.Emplist FROM Dept\_view d  
WHERE Deptno = 10) VALUES (1001, 'John Glenn', 10000);

# Trighere

```
insert into project_tab Values(10,10,10);
```

```
insert into Emp_tab values(1,'A','X',10,'10-JAN-2006',100,10,10);
```

```
insert into dept_tab values (10,'p1', 'iasi',1,2);
```

```
insert into manager_info values('A',1,40,30,1,11);
```

```
PRJ_LEVEL    PROJNO  RESP_DEPT
```

```
-----  
10           10           10  
1            11
```

```
-----  
1 A      X           10 10-JAN-06      100      10  
10
```

```
-----  
10 p1      iasi           1      2  
30           40
```

# Trighere

- Optiunea FOR EACH

```
CREATE TABLE Emp_log ( Emp_id NUMBER, Log_date DATE,  
New_salary NUMBER, Action VARCHAR2(20));
```

```
CREATE OR REPLACE TRIGGER Log_salary_increase
```

```
  AFTER UPDATE ON Emp_tab
```

```
  FOR EACH ROW
```

```
  WHEN (new.Sal > 1000)
```

```
  BEGIN
```

```
    INSERT INTO Emp_log (Emp_id, Log_date, New_salary, Action)  
    VALUES (:new.Empno, SYSDATE, :new.SAL, 'NEW SAL');
```

```
  END;
```

```
  /
```

```
UPDATE Emp_tab SET Sal = Sal + 1000 WHERE Deptno = 20;
```

# Trighere

- CREATE OR REPLACE TRIGGER Log\_emp\_update  
AFTER UPDATE ON Emp\_tab  
BEGIN  
    INSERT INTO Emp\_log (Log\_date, Action) VALUES (SYSDATE,  
    'Emp\_tab COMMISSIONS CHANGED');  
END;  
/  
• Accesul la valorile coloanelor in trigheri pentru linii.
  - Un trigger lansat de INSERT are acces numai la noile valori, vechile valori sunt considerate nule.
  - Un trigger lansat de UPDATE are acces la valoarea veche si noua pentru trigheri de linie cu BEFORE sau AFTER.
  - Un trigger lansat cu DELETE are acces numai la valorile vechi, noile valori sunt null. Nu putem modifica noile valori. Eroare ORA-4048.  
IF :new.Sal > 1000 ... IF :new.Sal < :old.Sal ...

# Trighere

- Modificarea coloanelor LOB cu un trigger:

```
drop table tab1;
```

```
create table tab1 (c1 clob);
```

```
insert into tab1 values ('Sir de caractere');
```

```
create or replace trigger trg1
```

```
before update on tab1
```

```
for each row begin
```

```
dbms_output.put_line('Old value of CLOB column: '||:OLD.c1);
```

```
dbms_output.put_line('Proposed new value of CLOB column:  
'||:NEW.c1);
```

```
/* Previously, we couldn't change the new value for a LOB. Now, we  
can replace it, or construct a new value using SUBSTR, INSTR...  
operations for a CLOB, or DBMS_LOB calls for a BLOB. */
```

```
:NEW.c1 := :NEW.c1 || to_clob('Alt sir de caractere');
```

```
dbms_output.put_line('Final value of CLOB column: '||:NEW.c1);
```

```
end; /
```

# Trighere

- set serveroutput on;  
update tab1 set c1 = 'Different Document ' ||  
'...Fragment</h1><p>Different text.';  
select \* from tab1;
- Detectarea operatiilor DML ce lanseaza un trigher:  
Daca una sau mai multe operatii DML(ON INSERT, ON DELETE,  
ON UPDATE), atunci trigherul poate folosi predicatele INSERTING,  
DELETING, UPDATING pentru a verifica ce tip de instructiune a  
lansat trigherul.  
IF INSERTING THEN ... END IF;  
IF UPDATING THEN ... END IF;  
IF DELETING THEN ... END IF;  
CREATE OR REPLACE TRIGGER ... .. UPDATE OF Sal, Comm  
ON Emp\_tab ...  
BEGIN ... IF UPDATING ('SAL') THEN ... END IF;  
END;

# Trighere

- Vizualizarea restrictiilor:

```
SELECT NAME, REFERENCED_OWNER,  
REFERENCED_NAME, REFERENCED_TYPE  
FROM ALL_DEPENDENCIES  
WHERE OWNER = 'SCOTT' and TYPE = 'TRIGGER';
```

---

|                      |                       |
|----------------------|-----------------------|
| OWNER                | NOT NULL VARCHAR2(30) |
| NAME                 | NOT NULL VARCHAR2(30) |
| TYPE                 | VARCHAR2(17)          |
| REFERENCED_OWNER     | VARCHAR2(30)          |
| REFERENCED_NAME      | VARCHAR2(64)          |
| REFERENCED_TYPE      | VARCHAR2(17)          |
| REFERENCED_LINK_NAME | VARCHAR2(128)         |
| DEPENDENCY_TYPE      | VARCHAR2(4)           |

# Trighere

- USER\_TRIGGERS
- ALL\_TRIGGERS
- DBA\_TRIGGERS

- RESTRICTII DE INTEGRITATE SI TRIGHERE:

```
CREATE OR REPLACE TRIGGER Dept_del_cascade
```

```
  AFTER DELETE ON Dept_tab
```

```
  FOR EACH ROW
```

```
  -- Before a row is deleted from Dept_tab, delete all
```

```
  -- rows from the Emp_tab table whose DEPTNO is the same as
```

```
  -- the DEPTNO being deleted from the Dept_tab table:
```

```
  BEGIN
```

```
  DELETE FROM Emp_tab WHERE Emp_tab.Deptno = :old.Deptno;
```

```
  END;
```

# Trighere

- Pentru restrictia de integritate referentiala:
  - Un trigger se defineste pentru tabela copil, ce va asigura faptul ca valorile cheii straine corespund valorilor cheii primare din tabela parinte.
  - Se pot defini mai multi triggeri pentru tabela parinte. Acestia garanteaza actiunea referentiala dorita (RESTRICT, CASCADE, SET NULL) pentru valorile cheii straine cind se modifica sau sterg chei in tabela parinte. In cazul INSERT in tabela parinte nu se cere nicio actiune.

```
CREATE OR REPLACE TRIGGER Emp_dept_check BEFORE
INSERT OR UPDATE OF Deptno ON Emp_tab FOR EACH ROW
WHEN (new.Deptno IS NOT NULL) -- Before a row is inserted, or
DEPTNO is updated in the Emp_tab -- table, fire this trigger to verify
that the new foreign -- key value (DEPTNO) is present in the
Dept_tab table. DECLARE Dummy INTEGER; -- to be used for
cursor fetch Invalid_department EXCEPTION; Valid_department
EXCEPTION; Mutating_table EXCEPTION; PRAGMA
EXCEPTION_INIT (Mutating_table, -4091);
```

# Trighere

*/\* Cursor used to verify parent key value exists. If present, lock parent key's row so it can't be deleted by another transaction until this transaction is committed or rolled back. \*/*

```
CURSOR Dummy_cursor (Dn NUMBER) IS SELECT Deptno FROM  
Dept_tab WHERE Deptno = Dn FOR UPDATE OF Deptno;
```

```
BEGIN
```

```
OPEN Dummy_cursor (:new.Deptno);
```

```
FETCH Dummy_cursor INTO Dummy; /* Verify parent key. If not  
found, raise user-specified error number and message. If found, close  
cursor before allowing triggering statement to complete:*/
```

```
IF Dummy_cursor%NOTFOUND THEN RAISE Invalid_department;  
ELSE RAISE valid_department;
```

```
END IF;
```

```
CLOSE Dummy_cursor;
```

# Trighere

## EXCEPTION

```
WHEN Invalid_department THEN
```

```
    CLOSE Dummy_cursor;
```

```
    Raise_application_error(-20000, 'Invalid Department' || ' Number' ||  
    TO_CHAR(:new.deptno));
```

```
WHEN Valid_department THEN CLOSE Dummy_cursor;
```

```
WHEN Mutating_table THEN NULL;
```

```
END;
```

- Trigher pentru tabela parinte(UPDATE si DELETE)

```
CREATE OR REPLACE TRIGGER Dept_restrict
```

```
    BEFORE DELETE OR UPDATE OF Deptno ON Dept_tab FOR  
    EACH ROW /* Before a row is deleted from Dept_tab or the primary  
    key (DEPTNO) of Dept_tab is updated, check for dependent foreign  
    key values in Emp_tab; rollback if any are found.*/
```

# Trighere

```
DECLARE Dummy INTEGER; -- to be used for cursor fetch  
    Employees_present EXCEPTION;
```

```
    employees_not_present EXCEPTION;
```

```
-- Cursor used to check for dependent foreign key values.
```

```
CURSOR Dummy_cursor (Dn NUMBER) IS SELECT Deptno FROM  
    Emp_tab WHERE Deptno = Dn;
```

```
BEGIN
```

```
    OPEN Dummy_cursor (:old.Deptno);
```

```
    FETCH Dummy_cursor INTO Dummy; /* If dependent foreign key is  
    found, raise user-specified error number and message. If not found,  
    close cursor before allowing triggering statement to complete*/.
```

```
    IF Dummy_cursor%FOUND THEN RAISE Employees_present;
```

```
    -- dependent rows exist
```

```
    ELSE RAISE Employees_not_present; -- no dependent rows
```

```
    END IF;
```

```
    CLOSE Dummy_cursor;
```

# Trighere

EXCEPTION

WHEN Employees\_present THEN

CLOSE Dummy\_cursor;

Raise\_application\_error(-20001, 'Employees Present in' || ' Department'  
|| TO\_CHAR(:old.DEPTNO));

WHEN Employees\_not\_present THEN

CLOSE Dummy\_cursor;

END;

- Trigheri UPDATE si DELETE SET NULL pentru tabela parinte.

CREATE OR REPLACE TRIGGER Dept\_set\_null AFTER DELETE  
OR UPDATE OF Deptno ON Dept\_tab

FOR EACH ROW

*/\*Before a row is deleted from Dept\_tab or the primary key (DEPTNO  
of Dept\_tab is updated, set all corresponding dependent foreign key  
values in Emp\_tab to NULL: \*/*

# Trighere

```
BEGIN
```

```
IF UPDATING AND :OLD.Deptno != :NEW.Deptno OR  
DELETING THEN
```

```
UPDATE Emp_tab SET Emp_tab.Deptno = NULL WHERE  
Emp_tab.Deptno = :old.Deptno;
```

```
END IF;
```

```
END;
```

- Trigher DELETE CASCADE pentru tabela parinte:

```
CREATE OR REPLACE TRIGGER Dept_del_cascade AFTER  
DELETE ON Dept_tab FOR EACH ROW
```

```
/*Before a row is deleted from Dept_tab, delete all rows from the  
Emp_tab table whose DEPTNO is the same as the DEPTNO being  
deleted from the Dept_tab table:*/
```

```
BEGIN
```

```
DELETE FROM Emp_tab WHERE Emp_tab.Deptno = :old.Deptno;  
END;
```

# Trighere

- Trigher UPDATE CASCADE pentru tabela parinte

*/\*Generate a sequence number to be used as a flag for -- determining if an update has occurred on a column:\*/*

```
CREATE SEQUENCE Update_sequence INCREMENT BY 1  
MAXVALUE 5000 CYCLE;
```

```
CREATE OR REPLACE PACKAGE Integritypackage AS Updateseq  
NUMBER;
```

```
END Integritypackage;
```

```
CREATE OR REPLACE PACKAGE BODY Integritypackage AS  
END Integritypackage;
```

*-- create flag col:*

```
ALTER TABLE Emp_tab ADD Update_id NUMBER;
```

```
CREATE OR REPLACE TRIGGER Dept_cascade1 BEFORE  
UPDATE OF Deptno ON Dept_tab DECLARE Dummy NUMBER;
```

# Trighere

```
/*Before updating the Dept_tab table (this is a statement trigger),  
generate a new sequence number and assign it to the public variable  
UPDATESEQ of a user-defined package named  
INTEGRITYPACKAGE: */
```

```
BEGIN
```

```
SELECT Update_sequence.NEXTVAL INTO Dummy FROM dual;  
Integritypackage.Updateseq := Dummy;
```

```
END;
```

```
CREATE OR REPLACE TRIGGER Dept_cascade2 AFTER  
DELETE OR UPDATE OF Deptno ON Dept_tab FOR EACH ROW  
/* For each department number in Dept_tab that is updated, cascade  
the update to dependent foreign keys in the Emp_tab table. Only  
cascade the update if the child row has not already been updated by  
this trigger: */
```

# Trighere

BEGIN

```
IF UPDATING THEN UPDATE Emp_tab SET Deptno =  
:new.Deptno, Update_id = Integritypackage.Updateseq --from 1st  
WHERE Emp_tab.Deptno = :old.Deptno AND Update_id IS NULL;  
/* only NULL if not updated by the 3rd trigger fired by this same  
triggering statement */
```

END IF;

IF DELETING THEN

```
/* Before a row is deleted from Dept_tab, delete all rows from the  
Emp_tab table whose DEPTNO is the same as the DEPTNO being  
deleted from the Dept_tab table:*/
```

```
DELETE FROM Emp_tab WHERE Emp_tab.Deptno = :old.Deptno;  
END IF; END;
```

```
CREATE OR REPLACE TRIGGER Dept_cascade3 AFTER  
UPDATE OF Deptno ON Dept_tab BEGIN UPDATE Emp_tab SET  
Update_id = NULL  
WHERE Update_id = Integritypackage.Updateseq; END;
```