12- TRIGGERS

Contenido

1. Què és un trigger

Un **TRIGGER** és un bloc PL/SQL associat a una acció (**event**): el bloc del trigger **s'executa automàticament** quan es genera l'event associat.

The trigger automatically fires (executes) whenever the triggering event occurs.

Els events poden ser:

- ✓ **events de dades** (*data event*): es genera un event de dades quan s'executa una sentència DML o DDL.
- ✓ **events de sistema**: es genera un event de sistema quan, per exemple, l'usuari connecta a la BD.

Els triggers estan sempre "*escoltant que passa a la BD*", esperant que s'executi la sentència DML/DDL associada.

Diferències entre triggers i procediments i funcions

- ✓ **Els triggers no es criden explícitament**, sinó que s'executen automàticament quan es genera l'event associat.
- ✓ Com no són cridats explícitament, **no poden rebre paràmetres**.
- ✓ Dintre del cos del trigger **no es permeten**: **COMMIT**, **SAVEPOINT** o **ROLLBACK**. Sí que es permeten als procediments.

GUIDELINES

- ✓ **El menor nombre de triggers possible**: L'ús excessiu de triggers fa **disminuir el rendiment** de la BD i pot crear **interdependències** difícils de mantenir.
- ✓ No utilitzar triggers per a realitzar accions que es poden fer fàcilment d'una altra forma. Per exemple, utilitzar restriccions per fer comprovacions sobre les dades, no fan falta triggers.
- ✓ **Triggers petits**: evitar tenir moltes sentències al cos del trigger. Millor crear un subprograma o un paquet i cridar-lo des del cos del trigger.

2. Triggers DML

Trigger associat a una sentència DML: INSERT, UPDATE o DELETE

Es poden classificar segons **quan** s'executa el trigger o **quants cops** s'executa.

SINTAXI

```
CREATE [OR REPLACE] TRIGGER trigger_name
TIMING EVENT1 [OR EVENT2 OR EVENT3] ON object_name
trigger_body
```

- ✓ **EVENT**: sentència DML que llença el trigger: `INSERT, UPDATE [OF column] i DELETE`.
- ✓ **object_name**: la taula o vista associada amb el trigger.
- ✓ **trigger_body**: accions que fa el trigger.
- ✓ **TIMING**: Indica quan s'executa el trigger en relació a la sentència que el dispara:
  - ➢ **BEFORE**: executa primer les accions del trigger i després la sentència associada.
  - ➢ **AFTER**: executa primer la sentència i després el trigger.
  - ➢ **INSTEAD OF**: executa el trigger enlloc de la sentència DML.

EXEMPLES DE TIMING (QUAN S'EXECUTA EL TRIGGER)

1. El trigger s'executa immediatament **abans** que s'actualitzi el salari d'un empleat:

```
CREATE OR REPLACE TRIGGER sal_upd_trigg
BEFORE UPDATE OF SALARY ON EMPLOYEES
BEGIN
    ...
END;
```

2. El trigger s'executa immediatament **després** que s'elimini un empleat:

```
CREATE OR REPLACE TRIGGER del_emp_trigg
AFTER DELETE ON EMPLOYEES
BEGIN
    ...
END;
```

3 . Trigger que s'executa quan s'actualitza **més d'una columna** (SALARY i COMISSION_PCT):

```
CREATE OR REPLACE TRIGGER sal_upd_trigg
BEFORE UPDATE OF SALARY, COMISSION_PCT ON EMPLOYEES
BEGIN
    ...
END;
```

4. Trigger que té més d'una event associat

```
CREATE OR REPLACE TRIGGER emp-DML_trigg
AFTER INSERT OR DELETE OR UPDATE ON EMPLOYEES
BEGIN
    ...
END;
```

3. Triggers de sentència i triggers de fila

Relacionat amb **quants cops s'executa el trigger**:

- ✓ **trigger de sentència (*statement-level trigger*)**: **s'executa només un cop per sentència**, encara que la sentència DML afecti a moltes files o no afecti a cap fila. És el tipus de trigger per defecte.
- ✓ **trigger de fila (row-level trigger): s'executa un cop per cada fila** afectada per la sentència del trigger. Utilitzat quan dintre del cos del trigger es necessita processar valors de les files afectades per la sentència DML.

4. Trigger de sentència: exemple1

Trigger que guarda l'usuari i la data cada cop que s'afegeix un empleat a la taula empleats.

```
CREATE OR REPLACE TRIGGER log_emp_changes
AFTER INSERT ON EMPLOYEES
BEGIN
    INSERT INTO log_emp_table(who, when)
    VALUES (USER, SYSDATE);
END;
```

Per provar si funciona el trigger:

1- Crear la taula **log_emp_table**:

```
CREATE TABLE log_emp_table (
    WHO VARCHAR2(30),
    WHEN DATE
)
```

2- Executar una sentència **INSERT**:

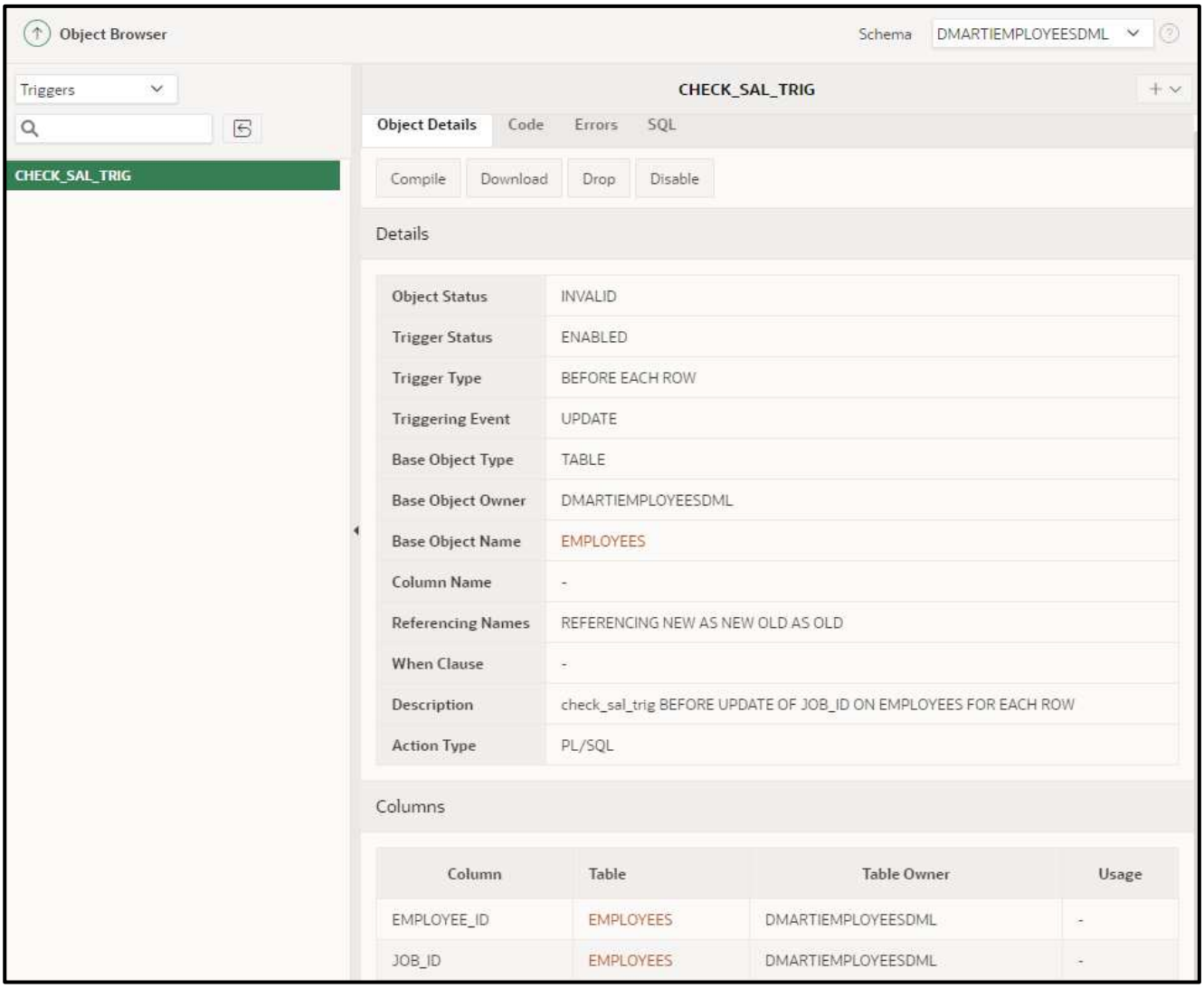
```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, LAST_NAME, EMAIL, HIRE_DATE, JOB_ID)
VALUES (500, 'Gómez', 'gomez@twit.com', SYSDATE, 'SA_MAN');
```

Si mirem el contingut de la taula **log_emp_table**:

WHO	WHEN
APEX_PUBLIC_USER	04/10/2020

El trigger ha afegit una fila amb l'usuari i la data de la inserció quan hem afegit un empleat a la taula empleats.

TRIGGERS I APEX

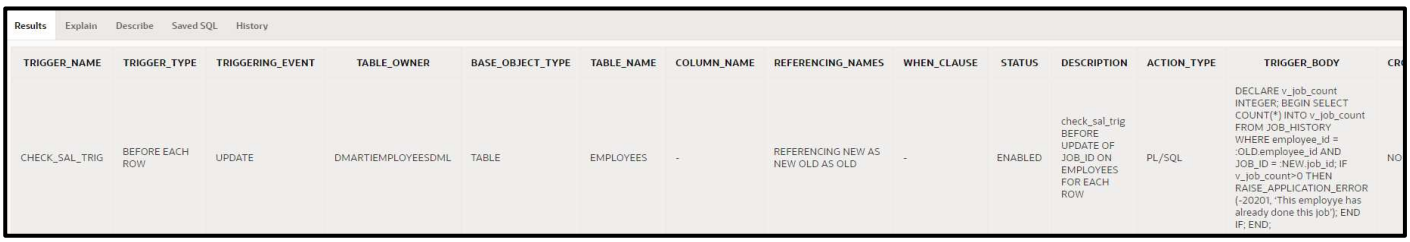


NOTA

Els triggers es guarden al diccionari de dades a la taula **USER_TRIGGERS**.

```
SELECT *
FROM USER_TRIGGERS
```

Mostra un munt d'informació del TRIGGER:



5. Trigger de sentència: exemple 2

Trigger que afegeix una entrada a la taula de login cada cop que s'executa amb èxit una sentència DML sobre la taula DEPARTAMENTS.

1- Crear la taula LOG_DEP_TABLE

```
CREATE TABLE log_dep_table (
    WHO VARCHAR2(30),
    WHEN DATE
)
```

2- Crear el trigger:

```
CREATE OR REPLACE TRIGGER log_dep_changes
AFTER INSERT OR UPDATE OR DELETE ON DEPARTMENTS
BEGIN
    INSERT INTO log_dep_table(who, when)
    VALUES (USER, SYSDATE);
END;
```

3- Executar sentències DML sobre la taula departaments

```
-- Afegeix un departament
INSERT INTO DEPARTMENTS (DEPARTMENT_ID,DEPARTMENT_NAME)
VALUES (666, 'Informàtica')

-- Canvia el nom del departament
UPDATE DEPARTMENTS
SET DEPARTMENT_NAME = 'INFOR.'
WHERE DEPARTMENT_ID=666;

-- Esborra el departament
DELETE FROM DEPARTMENTS
WHERE DEPARTMENT_ID=666
```

4- Mostrar contingut de la taula **LOG_DEP_TABLE**: el trigger ha afegit un entrada per cada operació DML.

WHO	WHEN
APEX_PUBLIC_USER	04/10/2020
APEX_PUBLIC_USER	04/10/2020
APEX_PUBLIC_USER	04/10/2020

6. Trigger de sentència: exemple3

Trigger que realitza comprovacions que no es poden fer amb les regles de restricció (constraint).

- ✓ Es permeten insercions a la taula EMPLOYEES de dilluns a divendres, però no es permeten en cap de setmana.
- ✓ Un intent d'afegir un EMPLOYEE en cap de setmana genera un missatge d'error.

Diu que es fa un ROLLBACK de l'INSERT, però crec que en realitat no és fa i ja està.

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON EMPLOYEES
BEGIN
    IF TO_CHAR(SYSDATE, 'DAY') IN ('SATURDAY', 'Sunday') THEN
         RAISE_APPLICATION_ERROR (-20500,
                  'En cap de setmana no es TREBALLA');
    END IF;
END;
```

EXERCICIS 1, 2

7. Predicats condicionals: DELETING, INSERTING, UPDATING

Suposem que tenim un trigger que es llença quan hi ha qualsevol sentència DML sobre la taula EMPLOYEES:

```
CREATE OR REPLACE TRIGGER log_emp_changes
BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYEES
BEGIN
    ...
END;
```

Ens interessa saber dintre del cos del trigger quina de les sentències DML (INSERT, UPDATE o DELETE) l'ha llençat, per actuar diferent en cada cas.

ORACLE utilitza les **variables booleanes DELETING**, **INSERTING** i **UPDATING** per indicar quina és l'operació que llença el trigger (TRUE).

```
CREATE OR REPLACE TRIGGER log_emp_changes
BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYEES
BEGIN
    IF DELETING THEN ...
    ELSIF INSERTING THEN ...
    ELSIF UPDATING THEN ...
END;
```

UPDATING es pot utilitzar per comprovar si s'actualitza una determinada columna:

```
CREATE OR REPLACE TRIGGER log_emp_changes
BEFORE UPDATE ON EMPLOYEES
BEGIN
    IF UPDATING('SALARY') THEN ...
    ELSIF UPDATING('JOB_ID') THEN ...
    ...
END;
```

8. Triggers de fila (ROW TRIGGERS)

Els triggers de sentència només s'executen un cop quan s'executa una sentència DML sobre la taula, independentment del nombre de files a les que afecti (1, vàries o cap).

Els triggers de fila (*row triggers*) **s'executen per cada fila** de la taula **afectada** per una sentència DML, justament abans de que la fila sigui "processada" (**BEFORE**) o justament després (**AFTER**).

Per declarar un trigger de sentència, s'afegeix **FOR EACH ROW**:

```
CREATE OR REPLACE TRIGGER log_emp_row
AFTER UPDATE OF SALARY ON EMPLOYEES FOR EACH ROW
BEGIN
    ...
END;
```

Qualificadors :OLD i :NEW

Des del trigger es pot fer referència als valors antics (**:OLD.NOM_COLUMNA**) i als nous (**:NEW.NOM_COLUMNA**) de les columnes que són actualitzades.

- ✓ Si la columna no canvia de valor, serà el mateix valor :OLD que :NEW.
- ✓ Compte que amb **INSERT** només tindrà valor el **:NEW** per que l'estem afegint i no hi ha :OLD !!!!

9. Exemple de triggers de fila

Es vol crear un trigger que guardi en una taula les modificacions de salaris dels empleats. A la taula es vol guardar l'id d'empleat, el salari antic, el salari nou i la data d'actualització.

1- Crear la taula log

```
CREATE TABLE LOG_EMP_SALARY_CHANGES
(   EMPLOYEE_ID NUMBER(6,0),
    OLD_SALARY NUMBER(8,2),
    NEW_SALARY NUMBER(8,2),
    CHANGE_DATE DATE
);
```

2- Crear el trigger

```
CREATE OR REPLACE TRIGGER log_emp_salary_changes
AFTER UPDATE OF SALARY ON EMPLOYEES FOR EACH ROW
BEGIN
    INSERT INTO LOG_EMP_SALARY_CHANGES
           (EMPLOYEE_ID, OLD_SALARY, NEW_SALARY, CHANGE_DATE)
    VALUES (:OLD.EMPLOYEE_ID, :OLD.SALARY, :NEW.SALARY, SYSDATE);
    DBMS_OUTPUT.PUT_LINE(':OLD.EMPLOYEE_ID: '  || :OLD.EMPLOYEE_ID ||
                         ' :NEW.EMPLOYEE_ID: ' || :NEW.EMPLOYEE_ID);
END;
```

3- Actualitzar els salaris dels empleats.

Actualitzem els salari dels empleats que tenen un id entre 200 i 330 (5 empleats)

```
UPDATE EMPLOYEES
SET SALARY = 1
WHERE EMPLOYEE_ID BETWEEN 200 AND 300
```

Sortida

```
:OLD.EMPLOYEE_ID: 200--- :NEW.EMPLOYEE_ID: 200
:OLD.EMPLOYEE_ID: 205--- :NEW.EMPLOYEE_ID: 205
:OLD.EMPLOYEE_ID: 206--- :NEW.EMPLOYEE_ID: 206
:OLD.EMPLOYEE_ID: 201--- :NEW.EMPLOYEE_ID: 201
:OLD.EMPLOYEE_ID: 202--- :NEW.EMPLOYEE_ID: 202

5 row(s) updated.
```

S'actualitzen 5 empleats.

El valor de l'id, com no canvia, és al mateix :OLD que :NEW.

Dades de la taula de log:

EMPLOYEE_ID	OLD_SALARY	NEW_SALARY	CHANGE_DATE
200	1	1	04/15/2020
205	12000	1	04/15/2020
206	8300	1	04/15/2020
201	13000	1	04/15/2020
202	6000	1	04/15/2020

5 rows returned in 0.03 seconds Download

10. Exemple de triggers de fila i qualificadors :OLD i :NEW

Creem dos triggers de files que mostrin quina és la sentència DML que s'executa sobre la taula EMPLOYEES: un trigger tindrà la clàusula before (**trigger_before**), l'altre la clàusula after (**trigger_after**).

En la primera versió, els triggers només mostren quina és la sentència DML que s'executa:

Versió 1

```
CREATE OR REPLACE TRIGGER trigger_before
BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYEES FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('TRIGGER BEFORE');

    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting');
    ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting');
    ELSIF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE('Updating');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER trigger_after
AFTER INSERT OR UPDATE OR DELETE ON EMPLOYEES FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('TRIGGER AFTER');
    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting');
    ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting');
    ELSIF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE('Updating');
    END IF;
END;
```

Afegim un empleat:

```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, HIRE_DATE, JOB_ID,
SALARY)
VALUES(1, 'Rafa', 'Nadal', 'rafa@nadal', '09/16/2022', 'IT_PROG', 10)
```

```
TRIGGER BEFORE
Inserting
TRIGGER AFTER
Inserting

1 row(s) inserted.
```

Els dos triggers mostren el missatge **Inserting**

Modifiquem el sou de l'empleat:

```
UPDATE EMPLOYEES
SET SALARY = 1000
WHERE EMPLOYEE_ID = 1
```

```
TRIGGER BEFORE
Updating
TRIGGER AFTER
Updating
:OLD.EMPLOYEE_ID: 1 :NEW.EMPLOYEE_ID: 1

1 row(s) updated.
```

Els dos triggers mostren el missatge **Updating**.

El trigger AFTER mostra a més els ids :NEW i :OLD de l'empleat modificat.

Eliminem l'empleat:

```
DELETE FROM EMPLOYEES
WHERE EMPLOYEE_ID = 1
```

```
TRIGGER BEFORE
Deleting
TRIGGER AFTER
Deleting

1 row(s) deleted.
```

Els dos triggers mostren el missatge Deleting.

Versió 2

Modifiquem els triggers per a que mostrin l'id **:NEW** i l'id **:OLD** que es crida al trigger.

```
CREATE OR REPLACE TRIGGER trigger_before
BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYEES FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('TRIGGER BEFORE');

    -- Mostra els ids de l'empleat
    DBMS_OUTPUT.PUT_LINE(':OLD.EMPLOYEE_ID: '  || :OLD.EMPLOYEE_ID ||
                         ' :NEW.EMPLOYEE_ID: ' || :NEW.EMPLOYEE_ID);
    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting');
    ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting');
    ELSIF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE('Updating');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER trigger_after
AFTER INSERT OR UPDATE OR DELETE ON EMPLOYEES FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('TRIGGER AFTER');

    -- Mostra els ids de l'empleat
    DBMS_OUTPUT.PUT_LINE(':OLD.EMPLOYEE_ID: '  || :OLD.EMPLOYEE_ID ||
                         ' :NEW.EMPLOYEE_ID: ' || :NEW.EMPLOYEE_ID);
    IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting');
    ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting');
    ELSIF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE('Updating');
    END IF;
END
```

Afegim un empleat:

```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, HIRE_DATE, JOB_ID,
SALARY)
VALUES(1, 'Rafa', 'Nadal', 'rafa@nadal', '09/16/2022', 'IT_PROG', 10)
```

```
TRIGGER BEFORE
:OLD.EMPLOYEE_ID:  :NEW.EMPLOYEE_ID: 1
Inserting
TRIGGER AFTER
:OLD.EMPLOYEE_ID:  :NEW.EMPLOYEE_ID: 1
Inserting

1 row(s) inserted.
```

:OLD.EMPLOYEE_ID és NULL: com afegim un empleat nou, no hi ha valors :OLD.

Molt important si s'ha d'utilitzar per fer una altra operació DML. No es podria utilitzar :OLD.EMPLOYEE_ID en un INSERT per què és NULL !!

Els valors **:OLD** són **NULL** si la sentència DML és un **INSERT**, independentment si el trigger és **before** o **after.**

Modifiquem el sou de l'empleat:

```
UPDATE EMPLOYEES
SET SALARY = 1000
WHERE EMPLOYEE_ID = 1
```

```
TRIGGER BEFORE
:OLD.EMPLOYEE_ID: 1 :NEW.EMPLOYEE_ID: 1
Updating
TRIGGER AFTER
:OLD.EMPLOYEE_ID: 1 :NEW.EMPLOYEE_ID: 1
Updating
:OLD.EMPLOYEE_ID: 1 :NEW.EMPLOYEE_ID: 1

1 row(s) updated.
```

Amb un UPDATE es pot accedir tant als valor :OLD com als :NEW.

Eliminem l'empleat:

```
DELETE FROM EMPLOYEES
WHERE EMPLOYEE_ID = 1
```

```
TRIGGER BEFORE
:OLD.EMPLOYEE_ID: 1 :NEW.EMPLOYEE_ID:
Deleting
TRIGGER AFTER
:OLD.EMPLOYEE_ID: 1 :NEW.EMPLOYEE_ID:
Deleting

1 row(s) deleted.
```

:NEW.EMPLOYEE_ID és NULL: com eliminem l'empleat, no hi ha valors :NEW.

Molt important si s'ha d'utilitzar per fer una altra operació DML. No es podria utilitzar :NEW.EMPLOYEE_ID en un INSERT per què és NULL !!

Els valors **:NEW** són **NULL** si la sentència DML és un **DELETE**, independentment si el trigger és **before** o **after.**

11. Clàusula REFERENCING: àlies per a :OLD i :NEW

Es pot donar un àlies per a :OLD i :NEW amb l'objectiu de millorar la claredat del trigger, quan pot haver-hi alguna confusió de noms.

EXEMPLE

Modifiquem el trigger de l'exemple anterior afegint una clàusula REFERENCING:

- ✓ :OLD -> :EMP_ANTIC
- ✓ :NEW-> :EMP_NOU

```
CREATE OR REPLACE TRIGGER log_emp_salary_changes
AFTER UPDATE OF SALARY ON EMPLOYEES
REFERENCING OLD AS EMP_ANTIC NEW AS EMP_NOU
FOR EACH ROW
BEGIN
    INSERT INTO LOG_EMP_SALARY_CHANGES
           (EMPLOYEE_ID, OLD_SALARY, NEW_SALARY, CHANGE_DATE)
    VALUES (:EMP_ANTIC.EMPLOYEE_ID, :EMP_ANTIC.SALARY,
            :EMP_NOU.SALARY, SYSDATE);
    DBMS_OUTPUT.PUT_LINE(':EMP_ANTIC.EMPLOYEE_ID: '  ||
                         :EMP_ANTIC.EMPLOYEE_ID ||
                        ' :EMP_NOU.EMPLOYEE_ID: ' ||
                         :EMP_NOU.EMPLOYEE_ID);
END;
```

EXERCICIS 3, 4