Les Triggers

HLIN511

Pascal Poncelet
LIRMM
Pascal.Poncelet@lirmm.fr
http://www.lirmm.fr/~poncelet

Présentation

- Un déclencheur est un traitement (sous forme de bloc PL/SQL) qui s'exécute automatiquement en réponse à un événement

- Deux types :
  - Déclencheur base de données
  - Déclencheur d'application
    - Rappel : les contraintes applicatives qui ont été définies lors de l'analyse de l'application

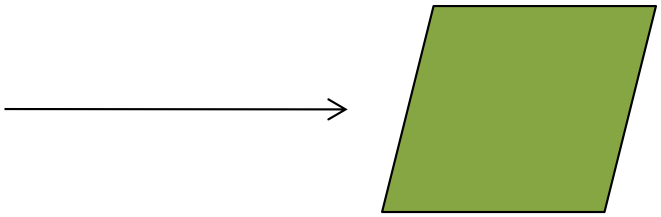
Exemple

SQL> **INSERT INTO** PILOTE

2 ……..

Plnum	Plnom	Adr	Sal
1	Dupond	Nice	15000
2	Dupré	Paris	20000
3	Duchamp	Toulouse	9000
…	…	…	…

Trigger Verif_Salaire



Syntaxe d’un trigger

CREATE [OR REPLACE] TRIGGER nom_trigger

{**BEFORE** | **AFTER | INSTEAD OF** } {**INSERT** [**OR**] | **UPDATE** [**OR**] | **DELETE**} [**OF** col_name] **ON** table_name [**REFERENCING OLD AS** o **NEW AS** n]

[**FOR EACH ROW**] **WHEN** (condition)

DECLARE Declaration-statements

BEGIN Executable-statements

EXCEPTION Exception-handling-statements

END;

DATE et SYSDATE

CREATE TABLE TESTDATE (LADATE **DATE**);

SELECT LADATE **FROM** TESTDATE;

LADATE

01-APR-16

DATE et SYSDATE

- **TO_CHAR** permet de convertir les dates

```
SELECT TO_CHAR(LADATE, 'YYYY/MM/DD') AS UNEDATE
 FROM TESTDATE;
```

```
UNEDATE
---------------
16-APR-16
```

DATE ET SYSDATE

- **TO_CHAR**(<date>, '<format>')
- Où format :
  - MM Mois en numérique (*e.g.*, 04)
  - MON Nom du mois en abrégé (*e.g.*, APR)
  - MONTH Nom du mois en entier (*e.g.*, APRIL)
  - DD Jour du mois (*e.g.*, 1)
  - DY Nom abrégé du jour (*e.g.*, FRI)
  - YYYY 4-digit de l'année (*e.g.*, 2016)
  - YY 2-digits de l'année (*e.g.*, 16)
  - RR Comme YY, mais les deux digits sont arrondis à l'année dans l'intervalle 1950 à 2049. Ansi 16 est considéré comme 2016 au lieu de 1016.
  - AM (or PM)Indicateur du méridien
  - HH Heure du jour (1-12)
  - HH24 Heure du jour (0-23)
  - MI Minute (0-59)
  - SS Seconde (0-59)

DATE et SYSDATE

- **TO_DATE** (chaine, '<format>')
- Opération inverse : conversion d'une chaîne en format DATE

INSERT INTO TESTDATE **VALUES**
(**TO_DATE**('2016/APR/02', 'yyyy/mm/dd');

- Où format est le même que **TO_CHAR**

DATE et SYSDATE

- SYSDATE permet de connaître la date système
- Peut être utilisé directement dans les triggers
- Par contre pour afficher la date système il faut utiliser une relation DUAL

SELECT TO_CHAR (**SYSDATE**, 'Jour DD-Mon-YYYY HH24') **AS** " Date Courante " **FROM** DUAL;

Date Courante

Lundi 21-Apr-2016 13:00:00

Exemple

```
CREATE OR REPLACE TRIGGER ctrl_mise_a_jour_employe
   BEFORE INSERT OR DELETE OR UPDATE ON EMPLOYES
DECLARE MESSAGE EXCEPTION;
BEGIN
   IF (TO_CHAR(SYSDATE,'DY')= 'SAM' OR TO_CHAR(SYSDATE,'DY')=
   'DIM')
      THEN RAISE MESSAGE;
   END IF;
   EXCEPTION
      WHEN MESSAGE THEN
      RAISE_APPLICATION_ERROR(-20324,'pas de mise à
            jour en fin de semaine');
END;
```

Règle de nommage

- Le nom d'un trigger doit être unique dans un même schéma

- Même s'il peut avoir le même nom qu'un autre objet (table, vue, procédure) il est préférable d'éviter pour ne pas avoir de conflit

Elément d'un trigger

- **BEFORE** : Le traitement est exécuté avant l'ordre LMD qui l'a déclenché
- **AFTER** : Le traitement est exécuté après l'ordre LMD qui l'a déclenché

- Ces options précisent le moment de l'exécution d'un trigger
- Remarque : les triggers **AFTER** row sont plus efficaces que les triggers **BEFORE** row car ils ne nécessitent qu'une seule lecture des données

Elément d'un trigger

- Autre élément de synchronisation
  - **INSTEAD OF** : Le traitement est exécuté en lieu et place de l'exécution de l'ordre LMD qui l'a déclenché
  - Utilisée souvent pour faire des mises à jour via des VUES

Elément d'un trigger

- Événement :
  - Indique quel ordre SQL déclenche le traitement :
    - **INSERT**
    - **UPDATE**
    - **DELETE**
    - Toute combinaison de ces ordres
  - Pour **UPDATE**, on peut avoir une liste de colonnes, le trigger ne se déclenche que si l'instruction **UPDATE** porte sur l'une au moins des colonnes précisée dans la liste
  - S'il n'y a pas de liste, le trigger est déclenché pour toute instruction **UPDATE** portant sur la table

Elément d'un trigger

- Table
  - La définition précise la table associée au trigger
  - Une et une seule table
  - Pas de vue (voir **INSTEAD OF**)

INSTEAD OF

```
CREATE VIEW les_clients AS
SELECT nom, prenom FROM CLIENT;

CREATE OR REPLACE TRIGGER insert_les_clients
INSTEAD OF INSERT ON les_clients
FOR EACH ROW
BEGIN
INSERT INTO CLIENT (num_client,nom,prenom) VALUES
   (seq_client.nextval,:new.nom,:new.prenom) ;
END;
```

Elément d’un trigger

- Type :
- Le type d’un trigger détermine :
  - Quand Oracle déclenche le trigger
  - Combien de fois le traitement doit s'exécuter suite à l'événement qui l'a déclenché
- Le type est défini par :

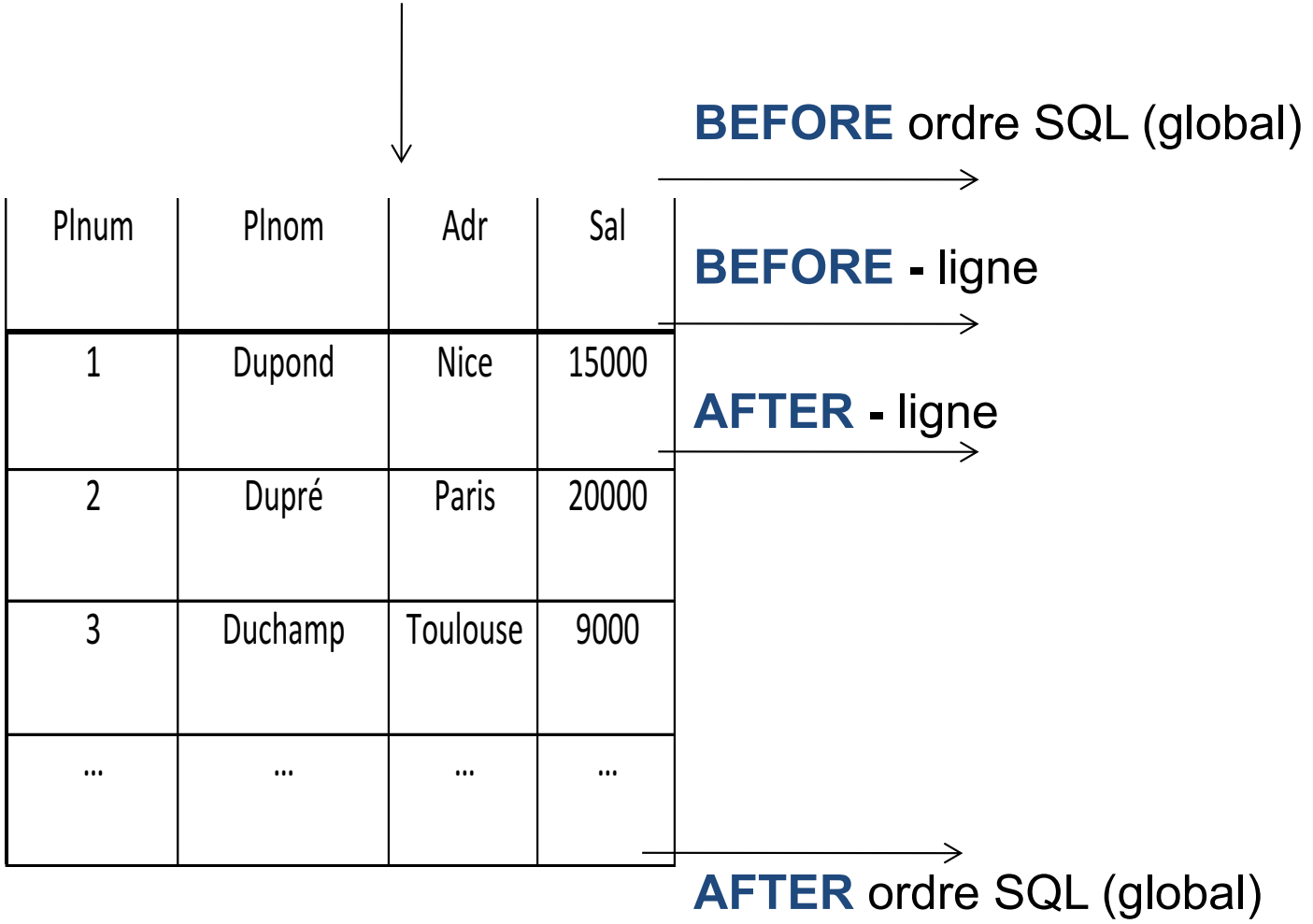
  BEFORE, AFTER, FOR EACH ROW

Les 2 types de triggers

- ORACLE propose deux types de triggers:
  - Les triggers lignes qui se déclenchent individuellement pour chaque ligne de la table affectée par le trigger
  - Les triggers globaux qui ne se déclenchent qu'un fois (option par défaut)
- Pour spécifier un trigger ligne : **FOR EACH ROW**

Exemple

SQL> **UPDATE** PILOTE **SET** sal=sal*1.1;



Plnum	Plnom	Adr	Sal
1	Dupond	Nice	15000
2	Dupré	Paris	20000
3	Duchamp	Toulouse	9000
…	…	…	…

Restrictions Triggers en ligne

- Il est possible d'ajouter une restriction sur les lignes via une expression logique SQL : c'est la clause **WHEN** :
  - Cette expression est évaluée pour chaque ligne affectée par le trigger
  - Le trigger n'est déclenché sur une ligne que si l'expression **WHEN** est vérifiée pour cette ligne
  - L'expression logique ne peut pas contenir une sous requête

WHEN (new.empno>0)

Empêche l'exécution du trigger si la nouvelle valeur de empno est 0, négative ou NULL

Elément d’un déclencheur

- Traitement - corps du déclencheur :
- Quelles actions à exécuter ?
  - Le corps du déclencheur est défini sous forme d'un bloc PL/SQL anonyme
  - Il peut contenir du SQL et du PL/SQL
  - Il est exécuté si l'instruction de déclenchement se produit et si la clause de restriction **WHEN**, le cas échéant, est évaluée à vrai.
  - Les corps d’un trigger ligne et d’un trigger global sont différents

Noms de corrélation

- Il est possible dans un trigger en ligne d'accéder à la nouvelle valeur et à l'ancienne (noms de corrélation)
- Attention :
  - Si l'instruction de déclenchement est **INSERT** seule la nouvelle valeur a un sens
  - Si l'instruction est **DELETE**, seule l'ancienne a un sens

Noms de corrélation

- La nouvelle valeur :

  :new.nom_colonne

- L'ancienne :

  :old.nom_colonne

IF :new.salaire > **:old**.salaire **THEN** …

REFERENCING

- Si une table s'appelle NEW ou OLD, il est possible d'utiliser **REFERENCING** pour éviter l'ambiguïté entre le nom de la table et le nom de corrélation

```
CREATE TRIGGER nomtrigger
BEFORE UPDATE ON new REFERENCING new AS autrenew
FOR EACH ROW
    BEGIN
      :autrenew.colon1:= TO_CHAR(:autrenew.colon2);
    END;
```

Les prédicats conditionnels

- Quand un trigger comporte plusieurs instructions de déclenchement (**INSERT** OR **DELETE** OR **UPDATE**), on peut utiliser des prédicats conditionnels (**INSERTING**, **DELETING** et **UPDATING**) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement

```
CREATE TRIGGER …
BEFORE INSERT OR UPDATE ON employe …….
BEGIN
   ……
   IF INSERTING THEN ……. END IF;
   IF UPDATING THEN …….. END IF; ……
END;
```

Nombre de triggers par table

- On peut avoir au maximum un trigger de chacun des types suivants pour chaque table :

  BEFORE UPDATE row
  BEFORE DELETE row
  BEFORE INSERT statement
  BEFORE INSERT row
  BEFORE UPDATE statement
  BEFORE DELETE statement
  AFTER UPDATE row
  AFTER DELETE row
  AFTER INSERT statement
  AFTER INSERT row
  AFTER UPDATE statement
  AFTER DELETE statement.

- Il ne peut y avoir qu'un **UPDATE** même si on change les noms de colonnes

Instructions SQL autorisées

- Autorisées : les instructions du LMD

- Interdites : les instructions du LDD et les instructions de contrôle des transactions (**ROLLBACK**, **COMMIT**)

Modification d’un trigger

CREATE OR **REPLACE** …

ou bien

DROP TRIGGER nom_trigger

Activation d'un trigger

- Un trigger est activé par défaut
- Désactivation d'un trigger :

  ALTER TRIGGER nomtrigger **DISABLE;**
- Pour désactiver tous les triggers associés à une table :

  ALTER TABLE nomtable **DISABLE ALL TRIGGERS;**
- Pour activer un trigger :

  ALTER TRIGGER nomtrigger **ENABLE;**
- Pour activer tous les triggers associés à une table :

  ALTER TABLE nomtable **ENABLE ALL TRIGGERS;**

Métabase

- Tables **USER_TRIGGERS, ALL_TRIGGERS** et **DBA_TRIGGERS**

Raise_application_error

- Procédure spécifique :

  raise_application_error (error_number,error_message)

  – error_number doit être un entier compris entre -20000 et -20999
  – error_message doit être une chaîne de 500 caractères maximum.
  – Quand cette procédure est appelée, elle termine le trigger, défait la transaction (**ROLLBACK**), renvoie un numéro d'erreur défini par l'utilisateur et un message à **l'application**

Exceptions

- Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites
- Possibilité de mettre dans exception dans un bloc PL/SQL

Exemple

```
CREATE OR REPLACE TRIGGER secure_emp
 BEFORE INSERT ON EMP
 BEGIN
  IF (TO_CHAR (SYSDATE,'DY') IN ('SAM', 'DIM'))
    OR (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN
     '08' AND '18'
  THEN RAISE_APPLICATION_ERROR (-20500,
            'Vous ne pouvez utiliser la table EMP
            que pendant les heures normales.');
 END IF;
 END;
/
```

Exemple

```
SQL> INSERT INTO emp (empno, ename, deptno)
  2  VALUES           (7777, 'DUPONT', 40);
INSERT INTO emp (empno, ename, deptno)
            *
ERROR at line 1:
ORA-20500: 'Vous ne pouvez utiliser la table EMP
   que pendant les heures normales.
ORA-06512: at "SCOTT.SECURE_EMP", line 4
ORA-04088: error during execution of trigger
'SCOTT.SECURE_EMP'
```

Exemple

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT OR UPDATE OR DELETE ON EMP
  BEGIN
   IF     (TO_CHAR (SYSDATE,'DY') IN ('SAM', 'DIM')) OR
   (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '08' AND '18') THEN
    IF DELETING THEN
     RAISE_APPLICATION_ERROR (-20502, 'Suppression impossible à cette heure.');
        ELSIF INSERTING THEN
           RAISE_APPLICATION_ERROR (-20500, 'Création impossible à cette
    heure.');
        ELSIF UPDATING ('SAL') THEN
           RAISE_APPLICATION_ERROR (-20503, 'Modification impossible à cette
    heure.');
        ELSE
           RAISE_APPLICATION_ERROR (-20504, 'Mises à jour impossibles à cette
    heure.');
      END IF;
  END IF;
```

Exemple

```
CREATE OR REPLACE TRIGGER check_salary_count
  AFTER UPDATE OF sal ON EMP
  DECLARE
    v_salary_changes NUMBER;
    v_max_changes    NUMBER;
  BEGIN
    SELECT upd, max_upd
    INTO   v_salary_changes, v_max_changes
    FROM   audit_table
    WHERE  user_name = user  AND    table_name = 'EMP'
    AND    column_name = 'SAL';
    IF v_salary_changes > v_max_changes THEN
      RAISE_APPLICATION_ERROR (-20501,  'Respectez le maximum : '||
       TO_CHAR (v_max_changes) || ' admissible pour le salaire');
   END IF;
  END;
/
```

Exemple

```
CREATE OR REPLACE TRIGGER audit_emp
  AFTER DELETE OR INSERT OR UPDATE ON emp
  FOR EACH ROW
  BEGIN
   IF DELETING THEN
      UPDATE audit_table SET del = del + 1
      WHERE  user_name = user AND table_name = 'EMP'
      AND    column_name IS NULL;
   ELSIF INSERTING THEN
     UPDATE audit_table SET ins = ins + 1
     WHERE  user_name = user AND table_name = 'EMP'
     AND    column_name IS NULL;
  ELSIF UPDATING ('SAL') THEN
     UPDATE audit_table SET upd = upd + 1
     WHERE  user_name = user AND table_name = 'EMP'
     AND    column_name = 'SAL';
  ELSE
     UPDATE audit_table SET upd = upd + 1
     WHERE  user_name = user AND table_name = 'EMP'
     AND    column_name IS NULL;
   END IF;
  END;
/
```

Exemple

```
CREATE OR REPLACE TRIGGER audit_emp_values
  AFTER DELETE OR INSERT OR UPDATE ON EMP
  FOR EACH ROW
  BEGIN
    INSERT INTO audit_emp_values (user_name,
     timestamp, id, old_last_name, new_last_name,
     old_title, new_title, old_salary, new_salary)
    VALUES (USER, SYSDATE, :old.empno, :old.ename,
     :new.ename, :old.job, :new.job, :old.sal, :new.sal);
  END;
  /
```

Exemple

```
CREATE OR REPLACE TRIGGER calcul_commission_pct
  BEFORE INSERT OR UPDATE OF sal ON EMP
  FOR EACH ROW
  WHEN (new.job = 'VENDEUR')
  BEGIN
   IF INSERTING THEN  :new.comm := 0;
    ELSE        /* Mise à jour du salaire */
     IF :old.comm IS NULL THEN
       :new.comm :=0;
    ELSE
      :new.comm := :old.comm * (:new.sal/:old.sal);
    END IF;
   END IF;
  END;
  /
```

Exemple

```
CREATE OR REPLACE TRIGGER cascade_updates
  AFTER UPDATE OF deptno ON DEPT
  FOR EACH ROW
  BEGIN
    UPDATE EMP
    SET    emp.deptno = :new.deptno
    WHERE  emp.deptno = :old.deptno;
  END;
  /
SQL>UPDATE DEPT
  2   SET     deptno = 1
  3   WHERE   deptno = 30;
*
ERROR at line 1:
ORA-04091: table DEPT is mutating, trigger/function
 may not see it
```

Exemple

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF sal, job ON EMP
  FOR EACH ROW
  WHEN (new.job <> 'PRESIDENT')
  DECLARE
     v_minsalary emp.sal%TYPE;
     v_maxsalary emp.sal%TYPE;
BEGIN
     SELECT MIN(sal), MAX(sal) INTO   v_minsalary, v_maxsalary
     FROM EMP WHERE  job = :new.job;
     IF :new.sal < v_minsalary OR  :new.sal > v_maxsalary THEN
        RAISE_APPLICATION_ERROR(-20505, 'salaire hors normes');
     END IF;
 END;
/
SQL> UPDATE EMP
  2  SET sal = 1500
  3  WHERE ename = 'DUPONT';
*
ERROR at line 2
ORA_4091 : Table EMP is mutating, trigger/function
may not see it
ORA_06512: at line 4
ORA_04088: error during execution of trigger
'check_salary'
```

Exemple

```
CREATE OR REPLACE TRIGGER check_salary
   BEFORE UPDATE OF sal ON emp
   FOR EACH ROW
   WHEN (new.sal < old.sal) OR  (new.sal > old.sal * 1.1)
   BEGIN
     RAISE_APPLICATION_ERROR (-20508,
     'Il ne faut pas diminuer le salaire ni
     l'augmenter de plus de 10%.');
  END;
/
```

Vérification de l'intégrité des données

Exemple

```
CREATE OR REPLACE TRIGGER cascade_updates
  AFTER UPDATE OF deptno ON DEPT
  FOR EACH ROW
  BEGIN
    UPDATE EMP
    SET    emp.deptno = :new.deptno
    WHERE  emp.deptno = :old.deptno;
  END;
 /
```

Vérification de l'intégrité référentielle

Exemple

```
CREATE OR REPLACE PROCEDURE increment_salaire
      (v_id IN DEPT.deptno%TYPE, v_salaire IN DEPT.total_salaire%TYPE) IS
   BEGIN
     UPDATE DEPT SET    total_sal = NVL (total_sal,0)+ v_salaire
     WHERE  deptno = v_id;
   END increment_salaire;
 /
CREATE OR REPLACE TRIGGER compute_salaire
   AFTER INSERT OR UPDATE OF sal OR DELETE ON EMP
   FOR EACH ROW
   BEGIN
   IF DELETING THEN increment_salaire(:old.deptno, -1 * :old.sal);
   ELSIF UPDATING THEN increment_salaire(:new.dept, :new.sal-:old.sal);
   ELSE /*insertion*/ increment_salaire(:new.deptno, :new.sal);
   END IF;
  END;
  /
```

Exemple

```
CREATE TRIGGER smic
    BEFORE INSERT OR UPDATE OF salaire ON EMP
FOR EACH ROW WHEN (new.salaire IS NULL) BEGIN
    SELECT 1000
    INTO :new.salaire
    FROM EMP;
END;
/
```

Ajouter une valeur de 1000 euros lorsque l'employé n'a pas de salaire

Exemple

```
CREATE TRIGGER verif_service
 BEFORE INSERT OR UPDATE OF numserv ON EMP
 FOR EACH ROW WHEN (new.numserv IS NOT NULL)
 DECLARE
    noserv INTEGER;
 BEGIN
        noserv:=0;
  SELECT numserv INTO noserv FROM SERVICE
  WHERE numserv=:new.numserv;
  IF (noserv=0)
        THEN raise_application_error(-20501, 'N° de service non correct');
  END IF;
END;
/
```

Vérification que le numéro du service de l'employé existe bien

Exemple

```
CREATE TRIGGER log
    AFTER INSERT OR UPDATE ON EMP
BEGIN
  INSERT INTO LOG(table, date, username, action) VALUES ('EMP',
    SYSDATE, SYS_CONTEXT ('USERENV',
     'CURRENT_USER'), 'INSERT/UPDATE ON EMP') ;
END ;
/
```

Sauvegarde dans un fichier log, la trace de la modification de la table Emp_tab (moment + utilisateur). N'est exécuté qu'une fois par modification de la table Emp_tab.

Exemple

```
CREATE OR REPLACE TRIGGER Print_salaire_changes
BEFORE 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 : ' || :old.sal || 'New : ’ || :new.sal || 'Difference : ' ||
     sal_diff);
END ;
```

Pour chaque modification (lignes mises à jour), le trigger va calculer puis afficher respectivement l’ancien salaire, le nouveau salaire et la différence entre ces deux salaires.

- Des questions ?