

2. Les déclencheurs

i. Définition :

Un trigger (ou déclencheur) est un bloc PL/SQL associé à une table (une et une seule) permettant de déclencher une action avant ou après un INSERT, UPDATE ou DELETE sur cette table. Les triggers sont stockés dans la base. Ils permettent de renforcer l'intégrité des données, d'auditer des actions sur une table ou bien calculer des valeurs dérivées pour d'autres colonnes. Les *Instructions SQL LMD (Insert, Update, Delete), LDD (Alter, Drop, ...) et les opérations systèmes (Logon, Logoff, Startup, Shutdown,..) sont autorisées. Par contre, les instructions de contrôle de transactions (ROLLBACK, COMMIT) ne sont pas autorisées.* La syntaxe d'un déclencheur est la suivante :

```
CREATE [OR REPLACE] TRIGGER nom_trigger {BEFORE|AFTER}
[INSERT OR] [UPDATE [OF nom_colonne] OR] [DELETE]
ON nom_Table
[FOR EACH ROW [WHEN (condition)]]
DECLARE
[variable declarations]
BEGIN
instructions
END;
```

ii. Manipulation des anciennes et nouvelles valeurs (:old et :new):

Il existe deux types de triggers :

- **Trigger sur ligne :** Le trigger est exécuté pour chaque ligne concernée par l'instruction insert, update ou delete (On utilise l'option "for each row").
- **Trigger sur instruction :** Le trigger est exécuté une seule fois pour l'instruction insert, update ou delete, même sur plusieurs lignes.

Pour les triggers de type "for each row", les colonnes de la ligne courante doivent être référencées spécifiquement selon que l'on veut l'ancienne ou la nouvelle valeur : (**:old.nom_colonne**, **:new.nom_colonne**). Exemples :

1- Le trigger suivant est un trigger simple affichant un message (OK) après l'insertion d'un étudiant :

```
Create or replace trigger monTrigger1
AFTER
INSERT on Etudiant
BEGIN
DBMS_OUTPUT.PUT_LINE ('OK');
END;
```



Test avec un insert:
 SQL> insert into Etudiant (enum) values ('E5');
 OK
 <-- affiché par le trigger
 1 row created. <-- affiché par Oracle SQLPlus

2- Le trigger suivant illustre un trigger de mise à jour. L'action du trigger est lancée pour la mise à jour d'une donnée quelconque de la relation Etudiant. Le trigger s'exécute à cause l'opération update vue dans son ensemble, et indépendamment de quels tuples touchés par la mise jour. :

```
Create or replace trigger monTrigger2
AFTER
UPDATE on Etudiant
BEGIN
DBMS_OUTPUT.PUT_LINE ('OK');
END;
```

Test avec un update:

```
SQL> update Etudiant SET Note_min=5 where Module='LANG';
OK
6 rows updated.
SQL> update Etudiant SET Semestre='S6' where Module='BDII';
OK
15 rows updated.
```

- 3- On pourrait souhaiter raffiner le déclenchement du trigger en indiquant quel est l'attribut concerné par une mise à jour. Dans l'exemple suivant : Seule la mise à jour sur la note minimale a lancé le trigger. Pas de message 'OK' à la deuxième requête update qui concerne le semestre.

```
Create or replace trigger monTrigger3
AFTER
UPDATE of Note_min on Etudiant
BEGIN
DBMS_OUTPUT.PUT_LINE ('OK');
END;
```

Test avec un update:

```
SQL> update Etudiant SET Note_min=5 where Module='LANG';
OK
6 rows updated.
SQL> update Etudiant SET Semestre='S6' where Module='BDII';
15 rows updated.
```

- 4- Dans l'exemple précédent, le trigger est lancé pour l'opération globalement (Trigger sur instruction). On peut souhaiter qu'un trigger exécute son code pour chaque tuple touché par la mise à jour. On utilise alors la clause : for each row (Trigger sur ligne)

```
Create or replace trigger monTrigger4
AFTER
UPDATE of Note_min on Etudiant
For EACH ROW
BEGIN
DBMS_OUTPUT.PUT_LINE ('OK');
END;
```

Test avec un update:

```
SQL> update Etudiant SET Note_min=5 where Module='LANG';
OK
OK
OK
OK
OK
OK
6 rows updated.
```

- 5- L'exemple suivant illustre l'utilisation des préfixes old et new. On peut accéder aux valeurs des champs touchés avant et après la mise à jour.

```
Create or replace trigger monTrigger5
AFTER
UPDATE of Note on Etudiant
For EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Nouveau:' ||: new. Note);
    DBMS_OUTPUT.PUT_LINE ('Ancien:' ||: old. Note);
END;
/
```

Test avec un update :

```
SQL> update Etudiant SET Note=NOTE+2 where Module='LANG';
Nouveau: 9
Ancien: 7
...
...
Nouveau: 12
Ancien: 10
6 rows updated.
```

- 6- La clause When : Cette clause permet de filtrer sur quels tuples sera exécuté le corps du trigger. Dans cet exemple, on n'affiche que les salaires inférieurs à 4.

Noter aussi la syntaxe old dans when sans (:)

```
Create or replace trigger monTrigger6
BEFORE
UPDATE of Note on Etudiant
For EACH ROW
when (old.Note < 4)
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Nouveau:' ||: new. Note);
    DBMS_OUTPUT.PUT_LINE ('Ancien:' ||: old. Note);
END;
```

```
SQL> update Etudiant SET Note=NOTE+2 where Module='LANG';
Nouveau: 5.
Ancien: 3
1 row updated
```

iii. Déclencheurs et RAISE APPLICATION_ERROR :

Soit le schéma relationnel suivant : *Employe (eid, enom, age, salaire)*, *Travail (#eid, #depid, pct_temps)* et *Departement (depid, budget, #responsable_id)*. Un employé peut travailler dans plus d'un département. Le champ pct_temps indique le pourcentage de temps pendant lequel un employé donné travaille dans un service donné. Nous illustrons dans les exemples suivants l'utilisation de RAISE_APPLICATION_ERROR en exprimant les exigences suivantes à l'aide des déclencheurs :

- a) Les employés doivent gagner un salaire minimum de 2000 DH :

```
CREATE OR REPLACE TRIGGER quest_1
BEFORE INSERT OR UPDATE ON Employe FOR EACH ROW
BEGIN
IF :NEW.salaire < 2000 THEN
RAISE_APPLICATION_ERROR(-20005, 'Les employés doivent gagner un salaire min 2000DH.');
END IF;
END;
```

- b) Le pourcentage total du temps de travail pour un employé doit être inférieur à 100%.

```
CREATE OR REPLACE TRIGGER quest_2
AFTER INSERT OR UPDATE ON Travail FOR EACH ROW
DECLARE
pct NUMBER;
BEGIN
SELECT SUM(pct_temps) INTO pct FROM Travail WHERE eid=:NEW.eid;
IF pct >=100 THEN
RAISE_APPLICATION_ERROR(-20005,'Le % total du temps de travail doit être < à 100');
END IF;
END;
```

- c) Un responsable doit toujours avoir un salaire plus élevé que n'importe quel employé qu'il dirige.

➔ (A faire... !)

- d) Chaque fois qu'un employé reçoit une augmentation, le salaire du responsable doit être augmenté du même montant.

➔ (A faire... !)

iv. Nombre de triggers AFTER et BEFORE par table

Pour chaque table, on peut avoir au maximum un trigger de chacun des types :

- 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.

Remarques :

- ✓ Même pour UPDATE, on ne peut pas en avoir plusieurs avec des noms de colonnes différents.
TRIGGERS EN CASCADE
- ✓ Un trigger peut provoquer le déclenchement d'un autre trigger.
- ✓ ORACLE autorise jusqu'à 32 triggers en cascade à un moment donné.
- ✓ AFTER ou BEFORE ?
 - Si le trigger doit déterminer si l'instruction DML est autorisée : utiliser BEFORE
 - Si le trigger doit "construire" la valeur d'une colonne pour pouvoir ensuite la mettre dans la table : utiliser BEFORE.
 - Si on a besoin que l'instruction DML soit terminée pour exécuter le corps du trigger : utiliser AFTER

v. Activation d'un déclencheur

Par défaut, un déclencheur est activé dès sa création. Il peut être ensuite activé ou désactivé. Au cas où un déclencheur est désactivé, ORACLE le stocke mais l'ignore.

Généralement, On peut désactiver un trigger si :

- * Il référence un objet non disponible ou bien
- * On veut charger un volume de données important ou bien
- * Charger des données déjà contrôlées.

✓ Pour Activer/ désactiver un trigger, on utilise l'instruction :

ALTER TRIGGER nomtrigger ENABLE /DISABLE ;

✓ On peut Activer/ Désactiver tous les triggers associés à une table avec :

ALTER TABLE nomtable ENABLE /DISABLE ALL TRIGGERS ;

vi. L'option REFERENCING et les prédictats conditionnels INSERTING, DELETING ET UPDATING

REFERENCING : Si une table s'appelle NEW ou OLD, on peut utiliser REFERENCING pour éviter l'ambiguïté entre le nom de la table et le nom de corrélation. Exemple :

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

INSERTING, DELETING ET UPDATING : Quand un trigger comporte plusieurs instructions de déclenchement (par exemple INSERT OR DELETE OR UPDATE), on peut utiliser des prédicts conditionnels (INSERTING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement. Exemple :

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

vii. Le déclencheur INSTEAD OF

Nous rappelons que la définition d'un déclencheur précise la table associée à ce déclencheur qui doit être une et une seule table (Non plus une vue).

Toutefois, Oracle offre la possibilité à l'utilisateur de créer un déclencheur sur une vue. En effet, le trigger INSTEAD OF permet de contourner l'interdiction de mise à jour au travers des vues.

Un déclencheur INSTEAD OF ne peut être défini que sur une vue et peut remplacer les insertions, les mises à jour et/ou les suppressions. Nous signalons aussi que ce type de déclencheur ne supporte pas les options BEFORE et AFTER.

- **Syntaxe :** Syntaxe d'un déclencheur déclaré INSTEAD OF INSERT (c'est-à-dire qu'il va s'exécuter lors d'une demande d'insertion dans la vue et va remplacer celle-ci).

```
CREATE OR REPLACE TRIGGER TRG_INSTEAD
INSTEAD OF INSERT ON NOM_VUE
DECLARE
...
BEGIN
...
...
END;
```

▪ Exemple

```
CREATE TABLE trainer ( full_name VARCHAR2(20) );
CREATE TABLE subject ( subject_name VARCHAR2(15) );
INSERT INTO trainer VALUES ('Charaf');
INSERT INTO subject VALUES ('Oracle');
CREATE VIEW Ma_vue AS
SELECT full_name, subject_name FROM trainer, subject,
```

Il s'agit d'une vue non modifiable que vous pouvez confirmer en exécutant une instruction DML sur celle-ci → Vous obtenez une erreur résultant d'une opération DML sur cette vue.

```
CREATE OR REPLACE TRIGGER Mon_déclencheur
INSTEAD OF INSERT ON Ma_vue FOR EACH ROW
BEGIN
INSERT INTO trainer (full_name) VALUES (:new.full_name);
INSERT INTO subject (subject_name) VALUES (:new.subject_name);
END
```

En cas d'exécution réussie, ce déclencheur insère une nouvelle ligne de données dans les deux tables sous-jacentes de la vue Ma_vue

viii. Exemples utiles de déclencheurs

a) Déclencheurs vs. Événements SYSTÈME

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER logon ON SCHEMA
BEGIN
INSERT INTO log_trig_table (user_id, log_date, action)
VALUES (user, sysdate, 'début de connexion');
END ;

CREATE OR REPLACE TRIGGER logoff_trig
AFTER logoff ON SCHEMA
BEGIN
INSERT INTO log_trig_table (user_id, log_date, action)
VALUES (user, sysdate, 'fin de connexion');
END ;
```

b) Déclencheurs vs. Sécurité

```
CREATE OR REPLACE TRIGGER Safe_Access
BEFORE INSERT OR UPDATE OR DELETE ON emp
BEGIN
IF
    TO_CHAR(sysdate, 'DY' IN ('SAT','SUN'))
THEN
    RAISE_APPLICATION_ERROR (-20505, 'Modification sur la table
                                EMP impossible le week-end');
END IF;
END;
```

c) Déclencheurs vs. Procédures

Un déclencheur peut appeler une procédure en utilisant l'instruction : CALL

```
CREATE TRIGGER test
    BEFORE INSERT ON EMP
BEGIN
    CALL log_execution;
END;
```

d) Déclencheurs vs. RéPLICATION

On considère les deux tables ci-dessous : TABLE1(clé col1 col2 col3) et TABLE2(clé col1 col2 col3)
On suppose que TABLE2 est la copie de TABLE1 sur un autre site.

Ecrire un déclencheur qui répercute toutes les mises à jour de TABLE1 sur TABLE2.

e) Déclencheurs vs. AUDIT

On considère la table T : T (clé col1 col2) et on lui adjoint une table d'audit : TA (opération datop).
Ecrire un déclencheur qui de sorte que toute mise à jour sur T entraîne l'enregistrement de l'opération (INSERT, UPDATE, DELETE) et son heure dans la table T1.