Les triggers d'Oracle

Création de triggers

- *Exemple :*

✓ **CREATE TRIGGER nom**
BEFORE DELETE OR INSERT OR UPDATE ON table
FOR EACH ROW
WHEN (new.empno>0)
DECLARE ..... <<<<déclarations>>>>
BEGIN
..... <<<< bloc d'instructions PL/SQL>>>>
END;

Le nom du trigger

- doit être unique dans un même schéma
- peut être le nom d'un autre objet (table, vue, procédure) mais à éviter

Option BEFORE/AFTER

- elle précise le moment de l'exécution du trigger
- les triggers AFTER row sont plus efficaces que les BEFORE row parce qu'ils ne nécessitent pas une double lecture des données.

Définition du trigger

- Elle comprend le type d'instruction SQL qui déclenche le trigger :

  DELETE, INSERT, UPDATE

  On peut en avoir une, deux ou les trois.

- Pour UPDATE, on peut spécifier une liste de colonnes. Dans ce cas, le trigger ne se déclenchera 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.

- ***La définition du trigger précise la table associée au trigger :***
  - ✓ **une et une seule table**
  - ✓ **pas une vue.**

Types de triggers

- Le type d'un trigger détermine :
  - ✓ quand ORACLE déclenche le trigger,
  - ✓ combien de fois ORACLE déclenche le trigger.
- Le type du trigger est défini par l'utilisation de l'une ou l'autre des options suivantes :

  BEFORE, AFTER, FOR EACH ROW

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 sont déclenchées une seule fois.
- Si l'option FOR EACH ROW est spécifiée, c'est un trigger ligne, sinon c'est un trigger global.

- **Pour les triggers lignes, on peut introduire une restriction sur les lignes à l'aide d'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-question.**
  - ✓ **Par exemple, WHEN (new.empno>0) empêchera l'exécution du trigger si la nouvelle valeur de EMPNO est 0, négative ou NULL.**

Le corps du trigger est un bloc PL/SQL :

- ✓ 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.
- ✓ Il est différent pour un trigger ligne et pour un trigger global.

Les noms de corrélation

- ✓ Dans un trigger ligne, on doit pouvoir accéder aux ancienne et nouvelle valeurs de colonne de la ligne.
- ✓ Les noms de corrélation permettent de désigner ces deux valeurs : un nom pour l'ancienne et un pour la nouvelle.
- ✓ Si l'instruction de déclenchement du trigger est INSERT, seule la nouvelle valeur a un sens.
- ✓ Si l'instruction de déclenchement du trigger est DELETE, seule l'ancienne valeur a un sens.

- ✓ **La nouvelle valeur est appelée**
  :new.colonne
- ✓ **L'ancienne valeur est appelée**
  :old.colonne
- ✓ ***Exemple :*** **IF :new.salaire < :old.salaire .....**
- ✓ **Si un trigger ligne BEFORE modifie la nouvelle valeur d'une colonne, un éventuel trigger ligne AFTER déclenché par la même instruction voit le changement effectué par le trigger BEFORE.**

L'option 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;
```

Les prédicats conditionnels INSERTING, DELETING et UPDATING

- ✓ **Quand un trigger comporte plusieurs instructions de déclenchement (par exemple 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.**
- ✓ ***Exemple :***

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

- **UPDATING peut être suivi d'un nom de colonne :**

```
CREATE TRIGGER ...
BEFORE UPDATE OF salaire, commission ON employe
.......
BEGIN
......
IF UPDATING ('salaire') 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.*
- ✓ Même pour UPDATE, on ne peut pas en avoir plusieurs avec des noms de colonnes différents.

Instructions SQL autorisées

- ✓ les instructions du LMD sont autorisées
- ✓ les instructions du LDD ne sont pas autorisées
- ✓ les instructions de contrôle de transaction (ROLLBACK, COMMIT) ne sont pas autorisées.

Ordre de traitement des lignes

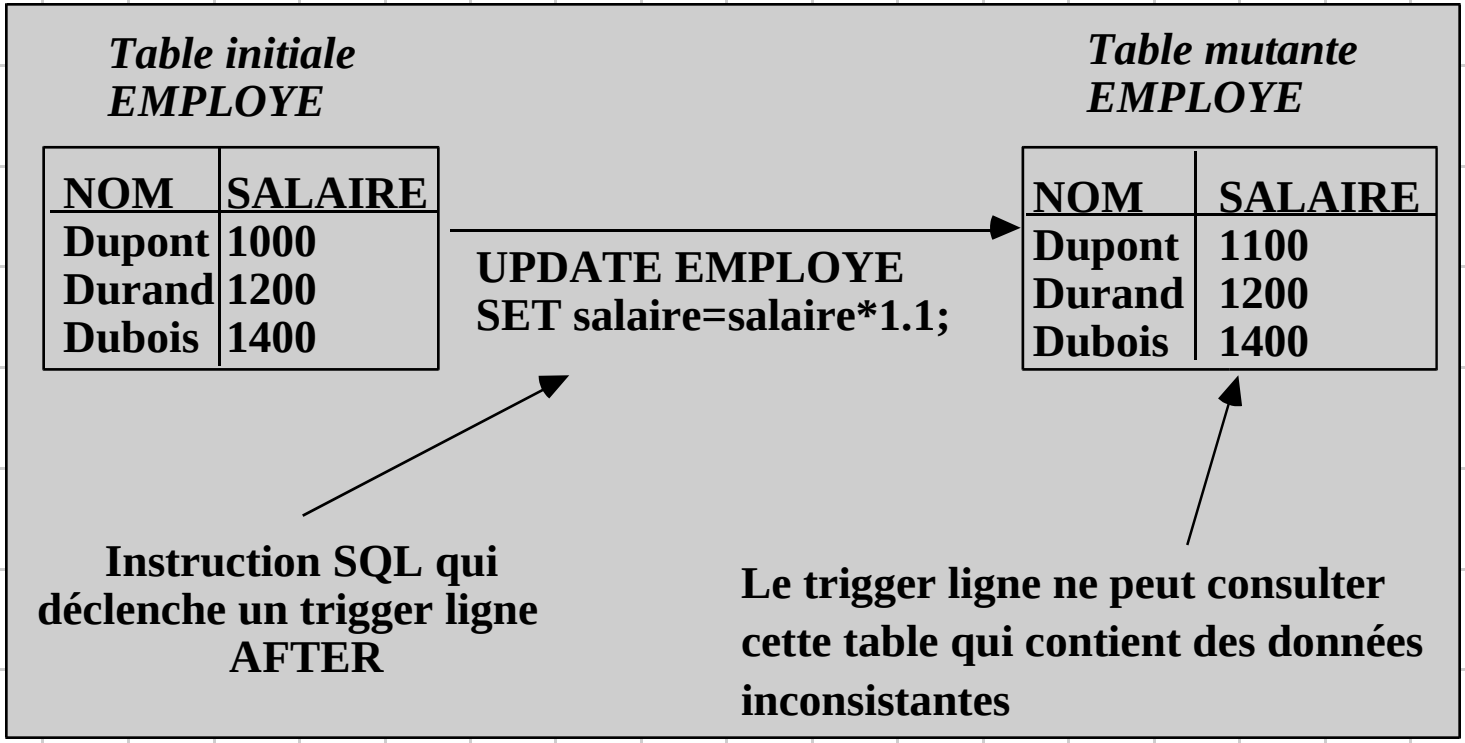
- ✓ On ne peut pas gérer l'ordre des lignes traitées par une instruction SQL.
- ✓ On ne peut donc pas créer un trigger qui dépende de l'ordre dans lequel les lignes sont traitées.

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.

Limite

✓ Un trigger ligne ne peut pas lire et/ou modifier la table concernée (appelée table mutante) par l'instruction (INSERT, UPDATE ou DELETE) qui a déclenché ce trigger. *Exemple :*



✓ On peut utiliser des tables temporaires.

Conditions nécessaires pour créer un trigger

- ✓ il faut avoir le privilège CREATE TRIGGER
- ✓ il faut soit posséder la table sur laquelle on veut définir un trigger, soit posséder le privilège ALTER sur la table sur laquelle on veut définir le trigger, soit posséder le privilège ALTER ANY TABLE

Modification de triggers

- ✓ Pour modifier un trigger, on refait une instruction CREATE TRIGGER suivie de OR REPLACE ou bien on supprime le trigger (DROP TRIGGER nomtrigger) et on le crée à nouveau.

Activation d'un trigger

- ✓ Un trigger peut être activé ou désactivé.
- ✓ S'il est désactivé, ORACLE le stocke mais l'ignore.
- ✓ On peut désactiver un trigger si :
  - ❊ il référence un objet non disponible
  - ❊ on veut charger rapidement un volume de données important ou recharger des données déjà contrôlées.
- ✓ Par défaut, un trigger est activé dès sa création.
- ✓ Pour désactiver un trigger, on utilise l'instruction *ALTER TRIGGER avec l'option DISABLE :*

  ALTER TRIGGER nomtrigger DISABLE;
- ✓ On peut désactiver tous les triggers associés à une table avec la commande :

  ALTER TABLE nomtable DISABLE ALL TRIGGERS;
- ✓ A l'inverse on peut réactiver un trigger :

  ALTER TRIGGER nomtrigger ENABLE;

  ou tous les triggers associés à une table :

  ALTER TABLE nomtable ENABLE ALL TRIGGERS;

Recherche d'information sur les triggers

- ✓ Les définitions des triggers sont stockées dans les tables de la métabase, notamment dans les tables USER_TRIGGERS, ALL_TRIGGERS et DBA_TRIGGERS

La procédure

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.

Gestion d'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.
- ✓ On peut introduire des exceptions en provoquant des erreurs.
  - Une exception est une erreur générée dans une procédure PL/SQL.
  - Elle peut être prédéfinie ou définie par l'utilisateur.
  - Un bloc PL/SQL peut contenir un bloc EXCEPTION gérant les différentes erreurs possibles avec des clauses WHEN.
  - Une clause WHEN OTHERS THEN ROLLBACK; gère le cas des erreurs non prévues.

Exceptions prédéfinies - quelques exemples

NO_DATA_FOUND

cette exception est générée quand un SELECT INTO ne retourne pas de lignes

DUP_VAL_ON_INDEX

tentative d'insertion d'une ligne avec une valeur déjà existante pour une colonne à index unique

ZERO_DIVIDE

division par zéro

etc

Quelques exemples

Quelques exemples

employe(numserv,....)
service(numserv,...)

```
/* vérifier que le service de l'employé existe bien */
CREATE TRIGGER verif_service
BEFORE INSERT OR UPDATE OF numserv ON employe
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;
```

employe(salaire,....)

```
/* mettre une valeur par défaut si le champ ne contient rien */
/* affecter 5000 au salaire d'un employe qui n'en a pas */
CREATE TRIGGER smic
BEFORE INSERT OR UPDATE OF salaire ON employe
FOR EACH ROW WHEN (new.salaire is null)
BEGIN
SELECT 5000
INTO :new.salaire
FROM  employe;
END;
```

employe(numemp,salaire,grade,...)
grille(grade,salmin,salmax)

```
/* vérifier le salaire d'un employé */
/* s'assurer que le salaire est compris dans les bornes
correspondant au grade de l'employé */

CREATE TRIGGER verif_grade_salaire
BEFORE INSERT OR UPDATE OF salaire, grade ON employe
FOR EACH ROW
DECLARE
minsal number;
maxsal number;
BEGIN
/* retrouver le salaire minimum et maximum du grade */
SELECT salmin,salmax
INTO minsal, maxsal
FROM grille
```

```
WHERE grade= :new.grade;
/* s'il y a un problème, on provoque une erreur */
IF (:new.salaire<minsal OR :new.salaire>maxsal)
THEN
raise_application_error (-20300,'Salaire'//TO_CHAR
   (:new.salaire)//
                    'incorrect pour ce grade');
EXCEPTION
WHEN no_data_found THEN
raise_application_error(-20301,'Grade incorrect');
END;
```

Conseils

- Utiliser des triggers pour garantir que, lorsqu'une opération spécifique est effectuée, les actions liées ont aussi réalisées.
- N'utiliser les triggers que pour les opérations globales, centralisées qui doivent être déclenchées indifféremment de l'utilisateur ou de l'application.
- N'utiliser des triggers que s'il n'y a pas d'autre possibilité intégrée dans Oracle (par exemple des contraintes déclaratives définies avec la table).
- Ne pas créer des triggers récursifs :
  - ✓ *par exemple, un trigger AFTER UPDATE sur une table qui ferait un UPDATE sur la même table.*
- Limiter la taille des triggers : à la première utilisation, le trigger est compilé (préférer une procédure stockée, déjà compilée).