SGBD-PL\SQL

(Procedural Language / Structured Query Language)

Chapitre 7 : Les déclencheurs

Faîçal Felhi

felhi_fayssal@yahoo.fr

Introduction

- On appelle déclencheur (trigger) un traitement qui se déclenche suite à un événement.
- Dans une programmation traditionnelle se faisait par un appel de sous programme.
- Il existe deux types de déclencheurs :
  - Des déclencheurs applicatifs crées et manipulés au niveau d'une application
  - Des déclencheurs de bases de données stockés dans le dictionnaire de données du SGBD et associés à des événements sur des tables

Utilité des déclencheurs

- Les déclencheurs sont utiles pour plusieurs fins :
  - Implémenter certaines règles métier du SI de l'organisation (par exemple, MAJ du stock suite à une entrée, déclenchement d'une demande de réapprovisionnement à la suite d'une insuffisance de stock,…)
  - Mettre en œuvre une politique de sécurité complexe (par exemple, interdire des modifications des données à certains utilisateurs, garder trace mises à jour douteuses,….

Caractéristiques des déclencheurs

- Les Triggers ressemblent aux sous programmes stockés sur les points suivants :
  - Sont des blocs PL/SQL nommés
  - Objets stockés dans Oracle sous forme de code source et p-code (exécutable)
- Mais ils en diffèrent sur les points suivants :
  - Liés à un événement : exécution implicite
  - Ne sont pas paramétrables
  - Peuvent être désactivés, etc …

Description d'un déclencheur

- Eléments de description

Un trigger est défini à travers les éléments suivants :

Elément	**Valeurs possibles**	**Explication**
Nom_Trigger	Choisie par le développeur	Identifiant du déclencheur
Nom_Table	Nom d'une table	La table sur laquelle le trigger est défini. Si l'événement est exécuté sur cette table, le trigger sera déclenché
Evénement	INSERT ou UPDATE, ou DELETE	Evénement déclenchant l'exécution du trigger : commande LMD ou une combinaison utilisant OR
Séquencement	BEFORE ou AFTER	Caractérise le séquencement du déclencheur (timing) par rapport à son événement. Avec BEFORE (AFTER), le déclencheur est lancé avant (après) que l'événement déclenchant ne soit exécuté.

Niveaux de déclenchement

Il existe deux niveaux de déclenchement :

1. Niveau instruction (statement level trigger)
2. Niveau ligne (row level trigger)

(1) : un déclencheur de niveau 1 s'exécute une seule fois

Création : la commande de création d'un déclencheur permet sa compilation et son stockage dans la métabase. Après une compilation correcte, le déclencheur est crée à l'état actif (ENABLE)

Création d'un TRIGGER

```
CREATE [ OR REPLACE ] TRIGGER nom_Trigger
  { BEFORE | AFTER }  événement
  ON nom_Table
DECLARE
-- déclarations variables, curseurs, records,…
BEGIN
- - traitement
EXCEPTION
- - Gestionnaires d'exceptions
END [nom_Trigger  ];
```

Déclencheur de Niveau ligne (row level trigger)

Un déclencheur de ligne s'exécute une fois pour chaque n-uplet affecté par l'événement déclenchant.

- Création d'un TRIGGER

CREATE [OR REPLACE] TRIGGER *nom_Trigger*

{ BEFORE | AFTER } *événement*

ON *nom_Table*

FOR EACH ROW [WHEN condition]

[REFERINCING {[old [AS] nom_old] | New [AS] nom_new]}]

DECLARE

-- *déclarations variables, curseurs, records,…*

BEGIN

- - traitement

EXCEPTION

- - Gestionnaires d'exceptions

END [*nom_Trigger*];

For each row	Clause optionnelle, si utilisée le déclencheur spécifié sera de niveau ligne. Autrement il est de niveau instruction
Condition	Option de la clause For each Row (valable uniquement pour les triggers de niveau ligne), introduit une condition selon laquelle le corps du déclencheur peut être exécuté ou simplement abandonné
Referencing	Permet de changer les noms standards des pseudo records OLD et NEW par de nouveaux noms nom_old et nom_new

- Exemples

1. Ecrire un trigger de BD qui affiche un message à la suite de la suppression d'un dept

Choix des composants du trigger				
Nom	Evénement	Séquencement	Niveau	Table
TRG_DEL_DEPT	DELETE	BEFORE (ou AFTER)	Ligne	DEPT

```
CREATE OR REPLACE TRIGGER TRG_DEL_DEPT
AFTER DELETE          -- ou bien BEFORE
ON DEPT
FOR EACH ROW          -- de niveau n-uplet
BEGIN
DBMS_OUTPUT.PUT_LINE('TRG0 : Dépt supprimé');
END TRG_DEL_DEPT;
```

Test :

```
SQL> delete from dept
  where deptno NOT IN (Select deptno from emp);
TRG0 : Dépt supprimé
1 ligne(s) supprimée(s).
```

Déclencheur de Niveau Instruction

- Remarques
  - Comme il s'agit d'un 'Row level Trigger' le corps du déclencheur sera exécuté avec chaque n-uplet affecté par l'événement déclenchant (DELETE). Si aucune ligne n'est affectée alors aucun message affiché
  - Utiliser un trigger de niveau instruction pour exécuter le corps du déclencheur une seule fois quelque soit le nombre de lignes affectées (même 0). Si aucune ligne n'est affectée alors le message sera affiché.

- Exemple de Trigger de niveau instruction

```
CREATE OR REPLACE TRIGGER TRG_DEL_DEPT_Niv_Ins
AFTER DELETE          -- ou bien BEFORE
ON DEPT
BEGIN
DBMS_OUTPUT.Put_Line('TRG_DEL_DEPT_Niv_Ins Déclenché');
END TRG_DEL_DEPT_Niv_Ins;
```

Test du trigger :

```
SQL> delete from dept where
   deptno is null;
TRG_DEL_DEPT_Niv_Ins Déclenché

0 ligne(s) supprimée(s).
```

Les Pseudo-records : old et : new

- Un trigger de niveau ligne se déclenche une fois avec chaque n-uplet traité par l'événement du trigger

- A l'intérieur d'un trigger de niveau ligne, on a souvent besoin d'accéder aux données du n-uplet en cours de manipulation. PL/SQL le permet par le biais des 2 pseudo-records **:old** et **:new** de même structure que la table sur laquelle le trigger est défini.

Evénement	OLD	NEW
INSERT	Tous les champs sont nulls	Valeurs à insérer en provenance de la commande INSERT qui a déclenché le trigger
UPDATE	Valeurs originales du n-uplet avant UPDATE	Nouvelles valeurs après UPDATE
DELETE	Valeurs Originales du n-uplet avant DELETE	

- Non valables pour les triggers de niveau instruction.
- Utilisables dans les curseurs implicites et explicites.

- Exemple

1. Ecrire un trigger qui enregistre dans la table ARCHIVE_DEPT chaque dept. Supprimé.

Nom	Evénement	Séquencement	Niveau	Table
TRG_Archive_Dept	DELETE	BEFORE	Ligne	DEPT

```
CREATE OR REPLACE TRIGGER TRG_Archive_Dept
BEFORE DELETE
ON DEPT
FOR EACH ROW
BEGIN
INSERT INTO ARCHIVE_DEPT (DEPTNO, Dname, Loc)
VALUES (:old.deptno, :old.dname, :old.loc);
EXCEPTION
        When OTHERS THEN
        RAISE_APPLICATION_ERROR(-20001,'Erreur
dans Trigger Before Delete On DEPT') ;
END TRG_Archive_Dept;
```

- **Exemple**

Ecrire un trigger de BD qui à la suite d'une modification de salaire enregistre dans la table AUDIT_SAL (Utilisateur, date_maj, Empno, anc_sal, nouv_sal)

```
CREATE OR REPLACE TRIGGER TRG_Audit_Sal
AFTER UPDATE OF SAL
ON EMP
FOR EACH ROW
BEGIN
INSERT INTO AUDIT_SAL
VALUES (user,sysdate, :old.empno, :old.sal, :new.sal);
EXCEPTION
    When others then null;
END ;
```

```
SQL>  update          emp
      set          sal= sal + 1000
      where deptno= 10;

3 ligne(s) mise(s) à jour.

SQL> select * from audit_sal;

UTILISATE  DATEMAJ    EMPNO  ANC_SAL  NOUV_SAL
-------- -------- --------- --------- ---------
SCOTT      18/04/02    7782              2450          3450
SCOTT      18/04/02    7839              5000          6000
SCOTT      18/04/02    7934              1300           2300
```

La clause WHEN «condition»

Cette clause est valide uniquement pour les row-level triggers. Si utilisée, la condition sera évaluée pour chaque ligne affectée par l'événement du trigger, et le corps du trigger n'est exécuté pour cette ligne que si l'expression logique est vraie.

- Les pseudo-record **old** et **new** sont autorisés dans la clause WHEN mais sans le préfixe (**:**)

- **Exemple 1**

```
CREATE OR REPLACE TRIGGER TRG_B_IU_EMP
BEFORE INSERT OR UPDATE of deptno
ON EMP
FOR EACH ROW
WHEN (new.deptno = 40)
BEGIN
   Update      dept
   Set         nb_emp = nvl(Nb_emp,0) + 1
   Where       deptno = :new.deptno;
END TRG_B_U_EMP;
```

Test du trigger :

```
SQL>        update      emp
        set     deptno      = 40
                where       deptno      = 10;

3 ligne(s) mise(s) à jour.
SQL> select * from dept where deptno = 40;
  DEPTNO DNAME          LOC            NB_EMP
-------- -------------- ------------ ---------
      40 OPERATIONS     BOSTON               3
```

- **Exemple 2**

Modification systématique de la commission pour la fonction SALESMAN : Augmenter leur commission en fonction de leur augmentation du salaire.

```
CREATE OR REPLACE TRIGGER TRG_B_Usal
BEFORE UPDATE OF SAL
ON EMP
FOR EACH ROW
WHEN (upper(new.job) = 'SALESMAN')
BEGIN
IF nvl(:old.sal,0)>0 THEN
:new.comm := :old.comm *(:new.sal/:old.sal);
END IF;
END TRG_B_Usal;
```

Test du trigger :

```
EMPNO       SAL      COMM                    JOB
--------- --------- ---------- ---------
7499              1600        300   SALESMAN

UPDATE                  EMP
   SET            SAL = SAL+320       -- 320 = 20% de sal
   WHERE     EMPNO = 7499;
EMPNO       SAL      COMM
--------- --------- ---------
    7499      1760       360
```

- **Remarques** :
  - La clause WHEN condition équivaut un test portant sur tout le corps du trigger
  - Ainsi, **TRG_B_UI_EMP** peut être réécrit comme suit :

```
CREATE OR REPLACE TRIGGER TRG_B_IU_EMP
BEFORE INSERT OR UPDATE of deptno
ON EMP
FOR EACH ROW
BEGIN
IF (:new.deptno = 40) THEN
   /* corps du trigger  */
END IF;
END TRG_B_U_EMP;
```

- **Exemple**

Interdire les modifications sur la table EMP en dehors des horaires de travail supposés entre 8h à 17h.

```
CREATE OR REPLACE TRIGGER TRG_Grant_Emp
BEFORE UPDATE
ON EMP
DECLARE
   E_ACCES_DENIED   EXCEPTION ;
BEGIN
If (to_number (to_char(sysdate,'HH24') ) NOT between 8 and 17)
THEN
RAISE E_ACCES_DENIED ;
End IF ;
EXCEPTION
WHEN  E_ACCES_DENIED  THEN
RAISE_APPLICATION_ERROR (-20001, 'Accès interdit en dehors des
   horaires de travail') ;
END;
```

Gestion des déclencheurs

- **DROP TRIGGER** triggerName **;**
- **Activation / Désactivation d'un trigger**

Parfois il est préférable de désactiver un trigger que de le supprimer en vue de le réactiver ultérieurement.

- **ALTER TRIGGER** triggerName **{ ENABLE |DISABLE } ;**

A la création un trigger est activé

Un trigger désactivé (DISABLE) continue à exister dans le DD.

- **Affichage des erreurs de compilation**

SQL>SHOW ERRORS

- **Activation/Désactivation de Tous les Triggers d'une table**

ALTER TABLE TableName **{ ENABLE | DISABLE } ALL TRIGGERS ;**

Quelques consignes PLSQL générales

- PL/SQL ne distingue pas entre majuscule et minuscules dans les identificateurs et les instructions
- Une variable déclarée par référence (%Type) à une colonne ayant la contrainte not null n'hérité pas cette contrainte
- Une seule déclaration par ligne est permise
- PL/SQL offre des conversions implicites . Ne jamais affecter une valeur textuelle dans une variable numérique, ceci conduit à un échec
- Les attributs des curseurs implicites sont utilisables comme des fonctions dans les instructions procédurales PL/SQL, il faut alors les affecter dans des variables si nécessaire
- Exception utilisateur (-20 OO0..-20 999)
- Others est une exception générique capable de capter toute erreur pour laquelle aucun gestionnaire n'a été prévu dans le bloc

Bibliographie

- « Oracle 11g: SQL, PL/SQL, SQL*Plus » de Jérôme Gabillaud
- « Oracle PL/SQL: guide du programmeur » de Steven Feuerstein et Bill Pribyl
- « Oracle 11g - Entraînez-vous sur SQL et PL/SQL » de Jérôme Gabillaud
- https://www.oracle.com/index.html