Quelques précisions concernant …

Les triggers en Oracle

1. Introduction

À la suite du laboratoire concernant les triggers sous Oracle, voici quelques précisions pour bien comprendre et préparer votre examen en base de données.

Les informations ci-dessous se basent sur la documentation officielle d'Oracle disponible sur le site http://www.oracle.com/pls/db102/homepage.

Lors de la correction du laboratoire, les informations décrites ci-dessous ont été prises en compte.

2. Les types de triggers

Il existe **deux** types de triggers différents : les triggers de table (STATEMENT) et les triggers de ligne (ROW). Quelle est la différence ?

- Les *triggers de table* sont exécutés **une seule fois** lorsque des modifications surviennent sur une table (même si ces modifications concernent plusieurs lignes de la table). Ils sont utiles si des *opérations de groupe doivent être réalisées* (comme le calcul d'une moyenne, d'une somme totale, d'un compteur, …). Pour des raisons de performance, il est préférable d'employer ces triggers plutôt que les triggers lignes.

- *Les triggers lignes* sont exécutés **« séparément » pour chaque ligne modifiée** dans la table. Ils sont très utiles s'il faut mesurer une *évolution pour certaines valeurs, effectuer des opérations pour chaque ligne en question*.

Lors de la création de triggers lignes, il est possible d'avoir accès à la valeur *ancienne* et la valeur *nouvelle* grâce aux mots clés *OLD* et *NEW*. Il n'est pas possible d'avoir accès à ces valeurs dans les triggers de table.

Exemple de trigger table :

```
CREATE TRIGGER log
AFTER INSERT OR UPDATE ON Emp_tab
BEGIN
     INSERT INTO log(table, date, username, action)
     VALUES ('Emp_tab', sysdate, sys_context('USERENV',
          'CURRENT_USER'), 'INSERT/UPDATE on Emp_tab') ;
END ;
```

Ce trigger table enregistre dans une table *log* la trace de la modification de la table Emp_tab. On mémorise ici le moment de la modification et l'utilisateur qui l'a provoqué. Il n'est donc exécuté qu'une seule fois par modification de la table Emp_tab.

3. La construction d'un trigger

3.1 Les conditions d'un trigger

Le trigger se déclenche lorsqu'un événement précis survient : BEFORE UPDATE, AFTER DELETE, AFTER INSERT, … Ces événements sont importants car ils définissent **le moment** d'exécution du trigger. Ainsi, lorsqu'un trigger BEFORE INSERT est programmé, il sera exécuté *juste avant* l'insertion d'un nouvel élément dans la table.

Pour rappel, lorsqu'un *trigger ligne* est mentionné à

INSERT	Pas d'accès à l'élément OLD (qui n'existe pas)
UPDATE	Accès possible à l'élément OLD et NEW
DELETE	Pas d'accès à l'élément NEW (qui n'existe plus)

3.2 Le code du trigger

Le trigger Oracle doit être écrit en PL/SQL. La forme d'un trigger est encore très dépendante du SGBD utilisé.

Exemple de trigger ligne (adapté de [1]):

```
CREATE OR REPLACE TRIGGER Print_salary_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 ;
```

Ce trigger est déclenché lorsque la table Emp_tab est mise à jour. 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. La condition précisée (WHEN new.Empno > 0) restreint le déclenchement du trigger (exécuté uniquement si new.Empno > 0)

Remarques :

- Un trigger ligne ne peut accéder à une table mutante. Dans [1], on peut lire : « *A mutating table is a table that is being modified by an* `UPDATE`*,* `DELETE` *or* `INSERT` *statement, or a table that might be updated by the effects of a* `DELETE CASCADE` *constraint. (…) The session that issued the triggering statement cannot query or modify a mutating table. This restriction prevents a trigger from seeing an inconsistent set of data. (…)* »

  Il faut donc comprendre qu'**il n'est pas permis** de consulter ou modifier une table *mutante*. Cette restriction préservera le trigger ligne de lire des données inconsistantes. Donc votre trigger ligne ne devrait jamais accéder

à la table sur laquelle il porte autrement que par OLD et NEW. Exemple (emprunté de [1]) :

```
CREATE OR REPLACE TRIGGER Emp_count
AFTER DELETE ON Emp_tab
FOR EACH ROW
DECLARE
     N INTEGER;
BEGIN
     SELECT COUNT(*) INTO n FROM Emp_tab;
     DBMS_OUTPUT.PUT_LINE('There are now ' || n ||
          'employees') ;
END ;
```

Dans cet exemple, la table mutante (celle qui est modifiée par DELETE) est Emp_tab. Or, une clause SELECT porte justement sur cette table. Le risque est d'obtenir un résultat complètement incohérent. Cette consultation est donc interdite.

- Dans un trigger ligne, il est possible de modifier les éléments de NEW. Ainsi, on peut lire dans [1] que : « *Old and new values are available in both `BEFORE` and `AFTER` row triggers. A new column value can be assigned in a `BEFORE` row trigger, but not in an `AFTER` row trigger (because the triggering statement takes effect before an `AFTER` row trigger is fired). If a `BEFORE` row trigger changes the value of `new.column`, then an `AFTER` row trigger fired by the same statement sees the change assigned by the `BEFORE` row trigger.* »

  Il est par conséquent autorisé de modifier les valeurs des colonnes au travers de NEW (et donc écrire dans le code PL/SQL `:new.val:=2` par exemple), uniquement dans des triggers lignes BEFORE.

- Si vous spécifier dans un trigger ligne que son déclenchement dépend *d'une colonne précise* dans une table (COMPTE OF CLIENT par exemple), il serait logique que l'élément OLD et NEW **ne porte que** sur cette colonne (COMPTE). Puisque c'est cet élément précis qui intervient dans votre trigger, il serait très étrange d'utiliser OLD ou NEW pour atteindre une autre colonne de la table (CLIENT dans notre exemple) même si le SGBD vous laisse faire.

3.3 La fin d'un trigger (les exceptions)

Parmi les limites d'un trigger, on peut lire dans [1] : « *DDL statements are not allowed in the body of a trigger. Also, no transaction control statements are allowed in a trigger. `ROLLBACK`, `COMMIT`, and `SAVEPOINT` cannot be used. (…)* »

Comme il n'est pas possible de définir des transactions ou de lancer un *ROLLBACK* pour terminer l'exécution normale, d'autres moyens doivent être mis en œuvre.

Ainsi, on peut lire dans [1] : « *Oracle Database allows user-defined errors in PL/SQL code to be handled so that user-specified error numbers and messages are returned to the client application. After received, the client application can handle the error based on the user-specified error number and message returned by Oracle Database.*

User-specified error messages are returned using the RAISE_APPLICATION_ERROR procedure. (…) This procedure stops procedure execution, rolls back any effects of the procedure, and returns a user-specified error number and message (unless the error is trapped by an exception handler). The error number must be in the range of -20000 to -20999

In database PL/SQL program units, an unhandled user-error condition or internal error condition that is not trapped by an appropriate exception handler causes the implicit rollback of the program unit.».

Par conséquent, dans un trigger BEFORE, vous pouvez empêcher la modification (INSERT, UPDATE, DELETE) en lançant une erreur via la procédure RAISE_APPLICATION_ERROR.

Exemple (adapté de [1]) :

```
CREATE OR REPLACE TRIGGER Emp_permit_changes
BEFORE INSERT OR DELETE OR UPDATE ON Emp_tab
DECLARE
     Not_on_weekends EXCEPTION;
BEGIN
     /* check for weekends: */
     IF (TO_CHAR(Sysdate, 'DY') = 'SAT' OR
          TO_CHAR(Sysdate, 'DY') = 'SUN') THEN
               RAISE Not_on_weekends;
     END IF;
EXCEPTION
     WHEN Not_on_weekends THEN
          Raise_application_error(-20324,'May not change '
               ||'employee table during the weekend');
END;
```

Bibliographie

[1] *Lance Ashdown et al.*, Oracle Database Application Developer's Guide – Fundamentals, 10g Release 2 (10.2), published by Oracle and available at http://www.oracle.com/pls/db102/homepage, 2005