Ressourcesinformatiques

Oracle 12c

SQL, PL/SQL, SQL*Plus

Jérôme GABILLAUD

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **RI12CORA** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1
Modèle relationnel

Chapitre 2
SQL

Chapitre 3
SQL*Plus et SQL Developer

Chapitre 4
PL/SQL

Chapitre 5
Objets de la base utilisant PL/SQL

Chapitre 6
Java

Chapitre 7

Le parseur XML

Chapitre 8
Application Express

Chapitre 5
Objets de la base utilisant PL/SQL

1. Introduction

En plus des blocs PL/SQL anonymes utilisés par SQL*Plus ou par les outils de développement (Oracle*Forms, Oracle*Reports...), on peut utiliser le PL/SQL dans des objets de la base, comme les procédures stockées (PROCEDURE, FUNCTION, PACKAGE) et les déclencheurs de bases de données.

2. Les déclencheurs de bases de données

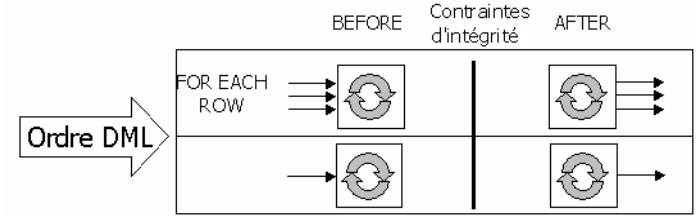
Un déclencheur ou trigger est un bloc PL/SQL associé à une table. Ce bloc s'exécutera lorsqu'une instruction du DML (INSERT, UPDATE, DELETE) sera demandée sur la table.

Le bloc PL/SQL qui constitue le trigger peut être exécuté avant ou après la mise à jour et donc avant ou après vérification des contraintes d'intégrité.

Les triggers offrent une solution procédurale pour définir des contraintes complexes ou qui prennent en compte des données issues de plusieurs lignes ou de plusieurs tables. Par exemple, pour garantir le fait qu'un client ne peut pas avoir plus de deux commandes non payées. Toutefois, les triggers ne doivent pas êtres utilisés lorsqu'il est possible de mettre en place une contrainte d'intégrité. En effet, les contraintes d'intégrité étant définies au niveau de la table et faisant partie de la structure de la table, la vérification du respect de ces contraintes est beaucoup plus rapide.

De plus, les contraintes d'intégrité garantissent que toutes les lignes contenues dans la tables respectent ces contraintes, tandis que les triggers ne prennent pas en compte les données déjà présentes dans la table lorsqu'ils sont définis.

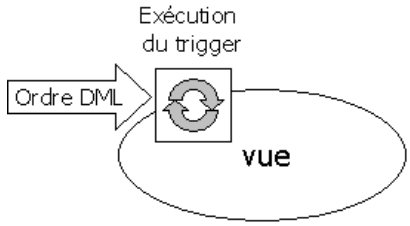
Le bloc PL/SQL associé au trigger peut être exécuté pour chaque ligne affectée par l'ordre DML (option FOR EACH ROW), ou bien une seule fois pour chaque commande DML exécutée (option par défaut).



Exécution avant ou après vérification des contraintes d'intégrité pour chaque ligne ou chaque ordre.

Dans les triggers BEFORE et FOR EACH ROW, il est possible de modifier les données qui vont être insérées dans la table pour qu'elles respectent les contraintes d'intégrité. Il est également possible de faire des requêtes de type SELECT sur la table sur laquelle porte l'ordre DML uniquement dans le cadre d'un trigger BEFORE INSERT. Toutes ces opérations sont impossibles dans les triggers AFTER car après vérification des contraintes d'intégrité, il n'est pas possible de modifier les données et comme la modification (ou ajout, ou suppression) de la ligne n'est pas terminée, il n'est pas possible d'effectuer des SELECT sur la table.

Il est également possible de poser des triggers sur les vues (VIEW) afin d'intercepter les ordres DML que l'on peut y exécuter. Ces triggers permettent de maîtriser toutes les opérations qui sont effectuées sur les vues et pour l'utilisateur final, la vue est en tout point similaire à une table puisqu'il peut y faire les opérations INSERT, UPDATE et DELETE. Ces triggers sont de type INSTEAD OF, c'est-à-dire que leur exécution va remplacer celle de la commande DML à laquelle ils sont associés. Ce type de trigger n'est définissable que sur les vues, et seul ce type de trigger peut être mis en place sur les vues.



Principe de fonctionnement des triggers instead of

Syntaxe

```
CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE/AFFTER/INSTEAD OF}
{INSERT/UPDATE[OF col,...]/DELETE}
ON nom_table [FOR EACH ROW]
[FOLLOWS nom_autre_trigger[,...]]
[ENABLE/DISABLE]
[WHEN (condition)]
Bloc PL/SQL
```

`OR REPLACE`

Remplace la description du trigger s'il existe déjà.

`BEFORE`

Le bloc PL/SQL est exécuté avant la vérification des contraintes de tables et la mise à jour des données dans la table.

`AFTER`

Le bloc PL/SQL est exécuté après la mise à jour des données dans la table.

`INSTEAD OF`

Le bloc PL/SQL qui suit remplace le traitement standard associé à l'instruction qui a déclenché le trigger (pour une vue uniquement).

`INSERT/UPDATE [OF col, ...]/DELETE`

Instruction associée au déclenchement du trigger. Plusieurs instructions peuvent déclencher le même trigger. Elles sont combinées par l'opérateur OR.

`FOR EACH ROW`

Le trigger s'exécute pour chaque ligne traitée par l'instruction associée.

`FOLLOWS nom_autre_trigger[,...]`

Oracle permet de définir plusieurs triggers pour la même table et le même événement. Dans ce cas, l'ordre relatif de déclenchement de ces triggers est indéterminé. Si l'ordre de déclenchement de ces triggers est important pour votre application, vous pouvez utiliser la clause FOLLOWS apparue en version 11. Cette clause permet d'indiquer que le trigger doit être déclenché après les triggers mentionnés.

`ENABLE/DISABLE`

Cette clause permet d'indiquer si le trigger est actif ou non dès sa création ; par défaut, un trigger nouvellement créé est actif. Créer un trigger désactivé permet de vérifier qu'il se compile correctement avant de le mettre réellement en service. Un trigger créé désactivé peut ensuite être activé par un ordre ALTER TRIGGER … ENABLE.

`WHEN (condition)`

La condition donnée doit être vérifiée pour que le code s'exécute.

Les données de la table à laquelle est associé le trigger sont inaccessibles depuis les instructions du bloc. Seule la ligne en cours de modification est accessible à l'aide de deux variables RECORD, OLD et NEW, qui reprennent la structure de la TABLE ou de la VIEW associée. Ces variables peuvent être utilisées dans la clause WHEN du trigger ou dans le bloc d'instructions. Dans ce dernier cas, elles sont référencées comme des variables hôtes avec le préfixe "**:**" (:OLD.nom_champs, :NEW.nom_champs).

Le terme OLD permet de connaître la ligne en cours de suppression dans un trigger DELETE ou la ligne avant modification dans un trigger UPDATE. Le terme NEW permet de connaître la nouvelle ligne insérée dans un trigger INSERT ou la ligne après modification dans un trigger UPDATE.

Les termes OLD et NEW sont fixés par défaut et il est possible d'utiliser d'autres termes en précisant la clause `REFERENCING OLD AS nouveau_nom NEW AS nouveau_nom`. Cette clause prend place juste avant la clause FOR EACH ROW (si elle existe) dans la définition du trigger.

Exemple

Exécution d'un bloc PL/SQL avant une suppression dans la table CLIENTS du user FLORIAN :

```
CREATE TRIGGER avant_sup_cli
      BEFORE    DELETE
      ON FLORIAN.CLIENTS
      DECLARE
         ...
      BEGIN
         ...
      END ;
```

Exécution d'un bloc PL/SQL après mise à jour de chaque ligne de la table ARTICLES si l'ancien prix est supérieur au nouveau :

```
create or replace trigger post_majprix
  after update of prix
  on articles
  for each row
  when (old.prix > new.prix)
declare
  ...
begin
  ...
end;
```

Pour chaque commande on souhaite connaître le nom de l'utilisateur Oracle qui a réalisé la saisie. La première étape consiste à ajouter une nouvelle colonne à la table des commandes. Cette colonne doit accepter la valeur NULL car pour les lignes de commande existantes, le nom de l'utilisateur Oracle est inconnu.

Modification de la table des commandes :

```
SQL> alter table commandes
  2    add (utilisateur varchar2(30));

Table modifiée.

SQL>
```

Dans le trigger, le nom de la nouvelle ligne est changé, il s'exécute avant vérification des contraintes d'intégrité pour chaque ligne insérée dans la table commandes.

Définition du trigger :

```
SQL> create or replace trigger bf_ins_commandes
  2    before insert
  3    on commandes
  4    referencing new as nouvelle
  5    for each row
  6  begin
  7    select user into :nouvelle.utilisateur from dual;
  8  end;
  9  /

Déclencheur créé.

SQL>
```

On souhaite connaître le nombre de commandes saisies par utilisateur Oracle. Pour éviter d'écrire une requête qui parcourt la totalité de la table des commandes, ce qui peut être très lourd, une table de statistiques va être alimentée par le trigger.

Création de la table de statistiques :

```
SQL> create table stat_util(
  2    utilisateur varchar2(30),
  3    nombre integer);

Table créée.

SQL>
```

Le trigger doit s'assurer que l'utilisateur existe dans la table des statistiques, et s'il n'est pas déjà présent alors il faut le créer. Le trigger s'exécute après chaque insertion de ligne, pour des raisons d'optimisation en cas de violation des contraintes d'intégrité.