PL/SQL : Stockage des traitements - 1

Généralités / Triggers

Est-ce bien à moi de faire tout cela ?

Sommaire

PL/SQL : Généralités

Généralités

- Déporter les traitements de manipulation de données vers le SGBD;
- Stockage du code sous forme de :
  - triggers,
  - fonctions/procédures,
  - packages;
- Langage propre : **PL/SQL**;
- Fournit un ensemble de fonctionnalités communes aux applicatifs/rôles.

Généralités

- Deux blocs principaux :
  - déclaration de variables (*DECLARE*, cf triggers),
  - exécution : *BEGIN ... END;* ;
- Gestion des exceptions;
- Structure générale :

```
<bloc de variables>
BEGIN
  <bloc de traitements>
EXCEPTION
  WHEN err1 THEN ...
  WHEN err2 THEN ...
  WHEN OTHERS THEN ...
END;
```

Sortie standard

- Package **DBMS_OUTPUT**
- Fonction *put_line(...)*

```
DBMS_OUTPUT.PUT_LINE("Coucou " || TO_CHAR(12));
```

- Activation de la sortie : **SET SERVEROUTPUT ON**

Variables

- Plusieurs types de variables :
  - scalaires (nombres, chaînes, date...),
  - composées,
  - curseurs ...
- Doc Oracle 11g
- Déclaration

```
-- prédéfini
v_prix NUMBER(6,2);
-- déduit
v_nom client.nom%TYPE;
```

Affectation

- **Attention !** Plusieurs formes d'affectation, soyez rigoureux !!
- Classique : :=

```
v_tva := 19.6;
```

- Sur requète : *INTO*

```
v_etudiant etudiant%rowtype;
...
SELECT * INTO v_etudiant FROM etudiant WHERE id=42;
```

- Sur curseur : *FETCH* (cf curseurs)

Types composés

- Type **RECORD**
- Déclaration :

```
-- le type
TYPE t_rec_eleve = RECORD (
  nom VARCHAR2(50),
  prenom VARCHAR2(30),
  age NUMBER(3)
);
...
-- la variable
v_eleve t_rec_eleve;

-- déduit d'une table
v_adresse adresse%rowtype;
```

Type tableau

- Deux types : **TABLE** (tableau associatif), **VARRAY** (tableau classique à taille variable). Le choix est fait selon le mode de parcours et la connaissance de la taille;
- Une dimension. Possibilité de construire des TABLE de TABLE ou VARRAY de VARRAY;
- Déclaration :

```
-- le type
TYPE tab_noms IS TABLE OF eleve.nom%type
INDEX BY PLS_INTEGER;

TYPE tab_jours IS VARRAY(7) OF VARCHAR2(8);
...
-- la variable
v_noms tab_noms;
v_jours tab_jours;
...
-- affectations
v_noms(3308) := 'toto';
v_noms(6712) := 'titi';
v_jours := tab_jours ('lundi', 'mardi', 'mercredi',
  'jeudi', 'vendredi', 'samedi', 'dimanche');
```

Instruction conditionnelle

```
IF cond1 THEN
  trt1
ELSIF cond2 THEN
  trt2
ELSIF cond3 THEN
  trt3
ELSE
  trt4
END IF;
```

Boucles

```
LOOP
  trt
  EXIT / EXIT WHEN condition
END LOOP;

FOR v_it IN deb..fin
LOOP
  trt
END LOOP;

WHILE condition
LOOP
  trt
END LOOP;
```

Parcours de tableau

- Parcourir un tableau associatif:

```
BEGIN
  ...
  i := v_noms.FIRST;
  WHILE i IS NOT NULL
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Élève ' || v_noms(i) || ', ID : ' || TO_CHAR(i) );
    i := v_noms.NEXT(i);
  END LOOP;
END;
```

```
BEGIN
  ...
  FOR i IN v_noms.FIRST..v_noms.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Élève ' || v_noms(i) || ', ID : ' || TO_CHAR(i) );
  END LOOP;
END;
```

Parcours de tableau

- Le premier élément d'un tableau "classique" possède l'indice **1** !
- La taille est donnée par : v_tab**.COUNT**
- Parcourir un tableau :

```
BEGIN
  ...
  FOR i IN 1..v_jours.COUNT
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_jours(i));
  END LOOP;
END;
```

Triggers et séquences

Triggers

- Un trigger est du code PL/SQL stocké en base et qui de *déclenche* lors d'un événement sur :
  - une TABLE,
  - une VUE,
  - un SCHÉMA/DATABASE
- Instructions de déclenchement parmi :
  - LDD : **CREATE, ALTER, DROP**
  - LMD : **INSERT, DELETE, UPDATE**
  - DATABASE : **STARTUP, SHUTDOWN, LOGON, ...**
- Moment de déclenchement autour de l'événement : **BEFORE / AFTER**
- *Conseil* : Il n'y a pas que les applicatifs qui modifient les données, pensez à réaliser des garde-fous pour les traitements stockés.

Triggers sur table

- Syntaxe :

```
CREATE [OR REPLACE] TRIGGER nom_trig
BEFORE|AFTER INSERT|DELETE|UPDATE ON nom_table
[FOR EACH ROW]
[WHEN condition]
DECLARE
variables
BEGIN
instructions PL/SQL
END;
```

- **FOR EACH ROW** : indique si les instructions s'appliquent sur toutes les lignes mises à jour. Deux variables sont instanciées : **:new** et **:old**, pour les nouveaux et anciens enregistrements;

- Lors de l'exécution d'un trigger, on ne requète pas la table en cours de mutation (a fortiori, on n'écrit pas la requête en cours d'exécution...).

Triggers sur table

- Plusieurs déclenchements :

```
CREATE [OR REPLACE] TRIGGER nom_trig
BEFORE INSERT OR DELETE ON nom_table
BEGIN
  IF INSERTING THEN
    ...
  ELSIF DELETING THEN
    ...
  END IF;
END;
```

- Colonne en particulier (UPDATE seulement) :

```
CREATE [OR REPLACE] TRIGGER nom_trig
BEFORE UPDATE OF col1, col2 ON nom_table
DECLARE
...
BEGIN
...
END;
```

Triggers sur table

- Un trigger possède deux états : **ENABLED / DISABLED**

```
ALTER TRIGGER nom_trig {ENABLED|DISABLED};
```

- S'il est modifié, un trigger se recompile :

```
ALTER TRIGGER nom_trig COMPILE;
```

- Suppression :

```
DROP TRIGGER nom_trig;
```

Triggers sur table

- Exemple : mise à jour de stock après commande. On considère ici que la vérification des quantités n'est pas à la charge de ce trigger.

```
CREATE OR REPLACE TRIGGER trig_maj_stock
AFTER INSERT ON ligne_commande
BEGIN
  :new.date_ajout := SYSDATE;
  UPDATE PRODUIT
  SET nb_stock = nb_stock - :new.quantite
  WHERE id_produit = :new.id_produit;
END;
```

- Exemple : on ne peut pas modifier si la nouvelle donnée est invalide
  Remarque : Il n'y a aucun lien avec l'exemple précédent, ce ne serait pas cohérent.... Pourquoi ?

```
CREATE OR REPLACE TRIGGER trig_maj_stock
BEFORE UPDATE OF nb_stock ON produit
FOR EACH ROW
BEGIN
  IF (:new.nb_stock < 0) THEN
    RAISE_APPLICATION_ERROR ( -20012, 'Mise à jour du stock impossible (<0)' ) ;
  END IF;
END;
```

Triggers sur vue

- Une vue accepte des modifications de données lorsqu'elle contient des clés/champs uniques, pas de regroupement ni de sous-requête...;
- Pour les autres, on peut associer un trigger à une vue : trigger **INSTEAD OF**;

```
CREATE [OR REPLACE] TRIGGER nom_trig
INSTEAD OF INSERT ON nom_vue
[DECLARE]
...
BEGIN
  traitement alternatif
END;
```

Triggers sur vue : Exemple

```
CREATE OR REPLACE VIEW vue_film_real AS
  SELECT num_film, titre, annee, f.num_ind as id_real, nom, prenom
  FROM film f, individu i
  WHERE f.NUM_IND = i.NUM_IND;
```

```
CREATE OR REPLACE TRIGGER trig_ins_vue_film_real
INSTEAD OF INSERT
ON vue_film_real
DECLARE
  v_id_real INDIVIDU.NUM_IND%type;
BEGIN
    BEGIN
      SELECT num_ind INTO v_id_real
      FROM INDIVIDU
      WHERE num_ind = :new.id_real;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        INSERT INTO individu VALUES (seq_id_real.NEXTVAL, :new.nom, :new.prenom)
        RETURNING num_ind INTO v_id_real;
    END;

    INSERT INTO film (num_film, titre, annee, num_ind)
    VALUES (seq_id_film.NEXTVAL, :new.titre, :new.annee,  v_id_real) ;

END ;
```

Séquence

- Permettent de générer "automatiquement" des valeurs entières (de clé en général);
- Le déclenchement d'un trigger, associé à une table permet la gestion d'auto-incrémentation;
- Elle peut être incrémentée et interrogée;

Séquence

- Syntaxe :

```
CREATE SEQUENCE nom_seq
[INCREMENT BY n] -- 1/défaut
[START WITH m] -- 1/défaut
[MINVALUE k]
[MAXVALUE p]
[CYCLE | NOCYCLE];
[CACHE l | NOCACHE];
...
```

- Accesseurs :
  - *nom_seq.NEXTVAL* : génère le nombre suivant,
  - *nom_seq.CURRVAL* : récupère la valeur en cours;
- Exemples

```
INSERT INTO users VALUES (seq_id_usr.NEXTVAL, "toto");
SELECT seq_id_usr.CURRVAL INTO valeurID FROM DUAL;
...
```

Séquence

- La gestion des clés primaires implique l'utilisation d'une séquence, dans un trigger, déclenché sur un événement d'insertion.

```
CREATE OR REPLACE TRIGGER id_eleve
BEFORE INSERT ON eleve
FOR EACH ROW
DECLARE
  v_user varchar2(10);
BEGIN
  -- nom de l'utilisateur connecté
  SELECT USER INTO v_user FROM DUAL;

  :new.id_eleve := seq_id_eleve.NEXTVAL;
  :new.date_ajout := SYSDATE;
  :new.cree_par := v_user;

END;
```

À vous de jouer !