Fondements des bases de données

Programmation en PL/SQL Oracle (2/2)

Équipe pédagogique BD

romuald.thion@univ-lyon1.fr

http://liris.cnrs.fr/~rthion/dokuwiki/enseignement:lif10/

Version du 18 février 2014

Les exceptions

Procédures et fonctions

Triggers

Les exceptions

Une exception est une erreur qui survient durant une exécution, elle est soit :

- prédéfinie par Oracle,
- définie par le programmeur.

Exceptions prédéfinies

`NO_DATA_FOUND` quand `SELECT ...INTO` ne retourne aucune ligne.

`TOO_MANY_ROWS` quand `SELECT ...INTO` retourne plusieurs lignes.

`VALUE_ERROR` érreur numérique.

`ZERO_DIVIDE` division par zéro

`OTHERS` toutes erreurs non interceptées.

Traitement des exceptions

Saisir une exception

- Une exception ne provoque pas nécessairement l'arrêt du programme : elle peut être saisie par une partie `EXCEPTION`.
- Une exception non saisie remonte dans la procédure appelante (où elle peut être saisie).

Exemple

```
BEGIN
   ...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
     ...
  WHEN TOO_MANY_ROWS THEN
     ...
  WHEN OTHERS THEN --optionnel
     ...
END;
```

Exceptions utilisateur

- Elles doivent être déclarées avec le type `EXCEPTION`
- On les lève avec l'instruction `RAISE`

Exemple

```
DECLARE
   v_salaire numeric(8,2);
   e_salaire_trop_bas EXCEPTION;
BEGIN
   SELECT sal INTO v_salaire
   FROM emp
   WHERE matr = 50;
   IF v_salaire < 2000 THEN
      RAISE e_salaire_trop_bas;
   END IF;
EXCEPTION
   WHEN e_salaire_trop_bas THEN
      dbms_output.put_line('Salaire trop bas');
   WHEN OTHERS THEN
      dbms_output.put_line(SQLERRM);
END;
```

Les exceptions

Procédures et fonctions

Triggers

Bloc anonyme ou nommé

- Un bloc anonyme PL/SQL est un bloc `DECLARE ...BEGIN ...END` comme dans les exemples précédents.
- On peut exécuter directement un bloc PL/SQL anonyme.
- On passe plutôt une procédure ou une fonction nommée pour réutiliser le code.

Procédure sans paramètre

Exemple

```
CREATE OR REPLACE PROCEDURE list_nom_emps IS
BEGIN
  DECLARE
    CURSOR c_nom_emps IS
      SELECT nom, adresse
      FROM emp;
  BEGIN
    FOR v_emp IN c_nom_emps LOOP
      dbms_output.put_line(
        'Client : ' || UPPER(v_emp.nom) ||
        ' Ville : ' || v_emp.adresse);
    END LOOP;
  END;
END;
/

CALL list_nom_emps();
```

Procédure avec paramètres

Exemple

```
CREATE OR REPLACE PROCEDURE
   liste_nom_emps(ville IN varchar2, v_result OUT number) IS
BEGIN
   BEGIN
        SELECT COUNT(*) INTO v_result
        FROM emp
        WHERE adresse LIKE ('%' || ville || '%');
   END;
END;
/

DECLARE
   v_result number;
BEGIN
   liste_nom_emps('Manchester',v_result);
   dbms_output.put_line(v_result);
END;
/
```

IN lecture seule

OUT écriture seule

IN OUT lecture et écriture

Fonctions sans paramètre

Exemple

```
CREATE OR REPLACE FUNCTION nb_emps
   RETURN NUMBER -- Type de retour
   IS
BEGIN
   DECLARE
      i NUMBER;
   BEGIN
      SELECT COUNT(*) INTO i
      FROM emp;
      RETURN i;
   END;
END;
/

SELECT nb_emps()
FROM DUAL;
```

DUAL est une pseudo table avec une seule colonne.

Fonctions avec paramètres

Exemple

```
CREATE OR REPLACE FUNCTION euro_to_fr(v_somme IN number)
   RETURN NUMBER
   IS
BEGIN
   DECLARE
      taux CONSTANT number := 6.55957;
   BEGIN
      RETURN v_somme * taux;
   END;
END;
/

SELECT euro_to_fr(15.24)
FROM dual;
```

Seuls les paramètres `IN` (en lecture seule) sont autorisés

Un peu plus ...

- Visualisation du résultat : `PRINT;`
- Description des paramètres : `DESC nom_procedure`
- Suppression de procédures ou fonctions :
  - `DROP PROCEDURE nom_procedure`
  - `DROP FUNCTION nom_fonction`
- Table système contenant les procédures et fonctions : `user_source`
- Les procédures et fonctions peuvent être utilisées dans d'autres procédures ou fonctions ou dans des blocs PL/SQL anonymes
- Les fonctions peuvent aussi être utilisées dans les requêtes

Les déclencheurs (triggers)

- Les contraintes prédéfinies ne sont pas toujours suffisantes
  Ex : *Tout nouveau prix d'un produit doit avoir une date de début supérieure à celle des autres prix pour ce produit*
- Exécuter des actions lors de certains événements :
  - `AFTER` ou `BEFORE`
  - `INSERT`, `DELETE` ou `UPDATE`
  - `FOR EACH ROW`
    - non (`STATEMENT`) : exécuté *une seule fois* pour la commande.
    - oui (`ROW`) : exécuté *à chaque ligne* concernée.

Syntaxe

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER} {INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name] ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
  --- sql statements
END;
```

Accès aux valeurs modifiées

Utilisation de NEW et OLD

- Si nous ajoutons un client dont le nom est `toto` alors nous récupérons ce nom grâce à la variable **:new.nom**
- Dans le cas de suppression ou modification, les anciennes valeurs sont dans la variable **:old.nom**

Exemple

Archiver le nom de l'utilisateur, la date et l'action effectuée dans une table `LOG_CLIENTS` lors de l'ajout d'un clients dans la table `CLIENTS`

- Créer la table `LOG_CLIENTS` avec la même structure que `CLIENTS`.
- Ajouter les colonnes `USERNAME`, `DATEMODIF`, `TYPEMODIF`.
- Créer un trigger `AFTER INSERT ON clients`

Accès aux valeurs modifiées

Exemple

```
CREATE or REPLACE TRIGGER logadd
   AFTER INSERT
   ON emp
   FOR EACH ROW
BEGIN
   INSERT INTO log_emp VALUES(
      :new.matr,
      ...
      USER,
      SYSDATE,
      'INSERT');
END;
/

INSERT INTO EMP VALUES (3,'Tarjan,␣Robert', 2446.13, 'Princet
SELECT * FROM log_emp;
```

Bug de sqldeveloper `Enter bindings` : exécuter le script *en intégralité* dans un nouveau fichier...

Prédicats conditionnels

- Lorsqu'un trigger a plusieurs opérations déclenchantes le corps peut avoir des prédicats conditionnels :
  - `IF INSERTING THEN ... END IF;`
  - `IF UPDATING THEN ... END IF;`
- On peut préciser les colonnes soumises aux opérations déclenchantes

Exemple

```
CREATE or replace TRIGGER t_emp_sal_increase
  BEFORE UPDATE OF sal, dep
  ON EMP
  FOR EACH ROW
BEGIN
  IF UPDATING('sal') THEN
    IF (:NEW.sal < :OLD.sal) THEN
      raise_application_error(-20000,'Sallary cannot decrease
    END IF;
  END IF;
END;
/
```

Important

Interdictions

- Les commandes de définition de données (LDD)
- les commandes de contrôle de transactions (`ROLLBACK`, `COMMIT`)

Ne doivent pas être utilisées dans le corps d'un trigger.

Remarques

- Pour éviter une modification de données dans un trigger `BEFORE`, il faut lever une exeception.
- Le dictionnaire de données a des vues sur les triggers : `USER_TRIGGERS`, `ALL_TRIGGERS`, `DBA_TRIGGERS`.
- La commande `DROP` permet de supprimer un trigger.

Tables mutantes et contraignantes

- Une table mutante est une table *en cours de modification* par une opération déclenchante (`UPDATE`, `DELETE`, `INSERT`) ou l'effet de `DELETE CASCADE` provenant de cette opération.
- Une table contraignante est une table qu'une *opération déclenchante doit lire*, soit directement via une commande SQL (`UPDATE SET ... WHERE`) ou indirectement pour une contrainte d'intégrité référentielle.

Les commandes SQL dans le corps d'un trigger ne peuvent pas

- Lire (par une requête) ou modifier un table mutante d'une opération déclenchante.
- Changer des valeurs sur les colonnes de clés (`PRIMARY`, `FOREIGN`, `UNIQUE`) d'une table contraignante.

Ces restrictions permettent d'éviter la consultation d'une table dans un état transitoire et donc incohérent.

Exemple d'erreur

```
CREATE OR REPLACE TRIGGER emp_count
  AFTER DELETE ON emp
  FOR EACH ROW
  DECLARE
    n integer;
  BEGIN
    SELECT COUNT(*) INTO n
    FROM emp;
    dbms_output.put_line(
      'On a ' || n || ' employes dans la base');
  END;

DELETE FROM emp WHERE matr = 0;
```

```
ORA-04091: table RTHION.EMP is mutating,
trigger/function may not see it.
```

Dans ce cas là, il ne faut pas utiliser `FOR EACH ROW` pour pouvoir déterminer la valeur du `SELECT COUNT(*) FROM emp;`.

Fin du sixième cours.