PL/SQL (ORACLE) - EXEMPLES

POLYTECH'MONTPELLIER - IG4

Un site plein d'informations :

`http://h50.isi.u-psud.fr/docmiage/oracle/doc/appdev.817/a77069/toc.htm`

Le but de PL/SQL est de fournir un environnement permettant d'allier SQL et des langages procéduraux en offrant des fonctionnalités de programmation impérative : gestion de variables, gestion des erreurs, fonctions, procédures, boucles, conditions, ...

Les programmes PL/SQL sont structurés en **blocs** correspondant à des unités logiques du programme (procédures, fonctions). Un bloc est composé de trois parties: la partie déclarative, la partie d'exécution, et la partie de gestion des erreurs. La syntaxe est la suivante :

```
[EN TETE]
[DECLARE
<constantes>
<variables>
<curseurs>
<exceptions utilisateurs>]
BEGIN
   <instructions>
   [EXCEPTION <gestion des exceptions>]
END ;
```

Exemple :

```
DECLARE
   qty_on_hand  NUMBER(5);
BEGIN
   SELECT quantity INTO qty_on_hand FROM inventory
      WHERE product = 'TENNIS RACKET'
      FOR UPDATE OF quantity;
   IF qty_on_hand > 0 THEN  -- check quantity
      UPDATE inventory SET quantity = quantity - 1
         WHERE product = 'TENNIS RACKET';
      INSERT INTO purchase_record
         VALUES ('Tennis racket purchased', SYSDATE);
   ELSE
      INSERT INTO purchase_record
         VALUES ('Out of tennis rackets', SYSDATE);
```

```
   END IF;
   COMMIT;
END;
```

L'en-tête permet de spécifier le type du bloc (procédure ou fonction). Elle est par exemple de la forme $CREATE\ OR\ REPLACE\ PROCEDURE\ toto$

1 Procédures et fonctions

La syntaxe pour une procédure est la suivante :

```
CREATE [OR REPLACE] PROCEDURE <nom procedure> [(liste des parametres)]
IS
<declarations>
BEGIN
   <instructions>
   [EXCEPTION
      <gestion des exceptions>  ]
END ;
```

La clause optionnelle $OR\ REPLACE$ permet de recréer une procédure existante.

Les clauses optionnelles IN (par défaut), OUT (une valeur peut être assignée au paramètre) et $IN\ OUT$ permettent de spécifier la manière dont sont utilisés les paramètres.

```
SQL> create or replace procedure coucou (n number)
  2  IS
  3  begin
  4  dbms_output.put_line('bonjour ' || to_char(n));
  5  end ;
  6  /

Procedure created.

SQL> execute coucou(3);
bonjour 3

PL/SQL procedure successfully completed.
```

Une fonction permet de retourner une valeur en sortie dont le type doit être spécifié :

```
SQL> CREATE FUNCTION AGEPLUS2 (n VARCHAR2) RETURN NUMBER
  2  IS
  3  an number;
  4  BEGIN
  5  SELECT annee_naissance into an from Artiste WHERE nom = n;
  6  return an + 2 ;
  7  END ;
  8  /

Function created.
```

La valeur retournée par la fonction doit être assignée à une variable SQL :

```
SQL> VARIABLE a number
SQL> select * from Artiste ;

NOM                  PRENOM          ANNEE_NAISSANCE
-------------------- --------------- ---------------
a                    toto                      1990

SQL> EXECUTE :a := AGEPLUS2('a');

PL/SQL procedure successfully completed.

SQL> print a;

         A
----------
      1992
```

Une procédure ou une fonction peut être effacée :

```
DROP PROCEDURE <nom procedure>
DROP FUNCTION <nom fonction>
```

2 Variables

Les variables peuvent être de tous les types SQL, ou du type $BOOLEAN$ (avec deux valeurs possibles : $TRUE$ et $FALSE$). La déclaration de variable comporte le nom de la variable, son type, et éventuellement une valeur d'initialisation. Par exemple on a :

```
part_no  NUMBER(4);
in_stock BOOLEAN;
credit_limit CONSTANT REAL := 5000.00;
```

Dans la partie d'instructions, l'affectation de valeurs aux variables est faite soit avec la syntaxe suivante :

```
tax := price * tax_rate;
```

soit par l'utilisation d'un résultat de requête :

```
SELECT sal * 0.10 INTO bonus FROM emp WHERE empno = emp_id;
```

soit par l'utilisation de fonctions existantes :

```
DECLARE
   my_sal REAL(7,2);
   PROCEDURE adjust_salary (emp_id INT, salary IN OUT REAL) IS ...
BEGIN
   SELECT AVG(sal) INTO my_sal FROM emp;
   adjust_salary(7788, my_sal); -- assigns a new value to my_sal
```

3 Variables de substitution

SQL permet l'utilisation de variables de substitution afin que des valeurs soient sollicitées de l'utilisateur :

```
SQL> select * from Artiste where nom = &no;
Enter value for no: 'a'
old   1: select * from Artiste where nom = &no
new   1: select * from Artiste where nom = 'a'

NOM                  PRENOM          ANNEE_NAISSANCE
-------------------- --------------- ---------------
a                    toto                        1990
```

ou

```
SQL> select * from Artiste where nom = '&no' ;
Enter value for no: a
old   1: select * from Artiste where nom = '&no'
new   1: select * from Artiste where nom = 'a'

NOM                  PRENOM          ANNEE_NAISSANCE
-------------------- --------------- ---------------
a                    toto                        1990
```

ou

```
SQL> select * from Artiste where &cond ;
Enter value for cond: nom = 'a'
old   1: select * from Artiste where &cond
new   1: select * from Artiste where nom = 'a'

NOM                  PRENOM          ANNEE_NAISSANCE
-------------------- --------------- ---------------
a                    toto                        1990
```

4 Curseurs

Les curseurs servent à manipuler l'information renvoyée par une requête. Ils sont définis avec la syntaxe suivante :

```
DECLARE
   CURSOR c1 IS
      SELECT empno, ename, job FROM emp WHERE deptno = 20;
```

On utilise alors les commandes $OPEN$, $FETCH$, et $CLOSE$. $OPEN$ exécute la requête associée au curseur, et positionne le curseur sur la première ligne du résultat. $FETCH$ renvoie la ligne courante et avance le curseur. Si on veut parcourir toutes les lignes, il faut donc faire une boucle contenant FETCH.

```
DECLARE
CURSOR c1 IS SELECT deptno, dname, locFROM dept;
dept_recc1%ROWTYPE;
BEGIN
OPEN c1;
LOOP
FETCH c1 INTO dept_rec;

END LOOP;
CLOSE c1 ;
END;
```

Quatre attributs permettent d'évaluer l'état du curseur :

- %NOTFOUND : vrai si le dernier fetchn'a ramen aucune ligne
- %FOUND (inverse de %NOTFOUND)
- %ROWCOUNT : compte le nombre de fetch excuts sur un curseur
- %ISOPEN : vrai si le curseur est ouvert

Il est possible de parcourir l'ensemble des lignes renvoyées grâce aux commandes $FORLOOP$

```
DECLARE
   CURSOR c1 IS
      SELECT ename, sal, hiredate, deptno FROM emp;
   ...
BEGIN
   FOR emp_rec IN c1 LOOP
      ...
      salary_total :=  salary_total + emp_rec.sal;
   END LOOP;
```

Les autres structures de contrôle sont les suivantes :

LOOP

```
LOOP
   -- instructions
END LOOP;
```

$WHILE...LOOP$

```
   WHILE salary <= 2500 LOOP
      SELECT sal, mgr, ename INTO salary, mgr_num, last_name
         FROM emp WHERE empno = mgr_num;
   END LOOP;
```

FOR ... LOOP

```
FOR i IN 1..order_qty LOOP
   UPDATE sales SET custno = customer_id
      WHERE serial_num = serial_num_seq.NEXTVAL;
END LOOP;
```

EXIT-WHEN permet de sortir de la boucle si une condition est satisfaite :

```
LOOP
   ...
   total := total + salary;
   EXIT WHEN total > 25000;  -- exit loop if condition is true
END LOOP;
```

IF..THEN...ELSE

```
IF job_title = 'SALESMAN' THEN
   amount := commission * 0.25;
ELSE
   amount := salary * 0.10;
END IF;
```

5 Packages

Les packages permettent de regrouper des types, variables, procédures, curseurs, ...

```
CREATE PACKAGE emp_actions AS  -- package specification
   PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...);
   PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;

CREATE PACKAGE BODY emp_actions AS  -- package body
   PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...) IS
   BEGIN
      INSERT INTO emp VALUES (empno, ename, ...);
   END hire_employee;
   PROCEDURE fire_employee (emp_id NUMBER) IS
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
   END fire_employee;
END emp_actions;
```

6 Exceptions

Il est possible de déclarer des exceptions dans la partie déclaration pour ensuite lever les exceptions apparaissant lors des instructions et faire une action spécifique.

```
DECLARE
   ...
   comm_missing EXCEPTION;  -- declare exception
BEGIN
   ...
   IF commission IS NULL THEN
      RAISE comm_missing;  -- raise exception
   END IF;
   bonus := (salary * 0.10) + (commission * 0.15);
EXCEPTION
   WHEN comm_missing THEN ... -- process the exception
```

Certaines exceptions sont prédéfinies : $CURSOR\_ALREADY\_OPEN$, $INVALID\_CURSOR$, NO_DATA_FOUND, TOO_MANY_ROWS, $ZERO\_DIVIDE$, ...

On a alors :

```
...
EXCEPTION
   WHEN NO_DATA_FOUND OR ZERO_DIVIDE THEN ...
   WHEN OTHERS THEN ...
```

$OTHERS$ permet de gérer tous les autres cas.

7 Entrées/sorties

Le module $DBMS\_OUTPUT$ permet d'afficher des informations :

```
  DBMS_OUTPUT.ENABLE              (autorise l'affichage)
  DBMS_OUTPUT.DISABLE             (interdit l'affichage)
  DBMS_OUTPUT.PUT(chaine)         (affiche la chaine)
  DBMS_OUTPUT.PUT_LINE(chaine)    (affiche la chaine et passe a la ligne)
  DBMS_OUTPUT.NEW_LINE            (passe a la ligne)
```

Exemple :

```
SQL> SET SERVEROUTPUT ON
SQL> create or replace procedure coucou
  2  IS
  3  an number;
  4  BEGIN
  5  select annee_naissance INTO an from Artiste ;
  6  dbms_output.put_line('annee ' || to_char(an));
  7  END ;
  8  /

Procedure created.
```

```
SQL> execute coucou ;
annee 1990

PL/SQL procedure successfully completed.
```

noindent N'oubliez pas le $SETSERVEROUTPUTON$!

7.1 Déclencheurs (*triggers*)

Les déclencheurs permettent de gérer les contraintes d'intégrité. Ils peuvent intervenir AVANT ou APRES les événements suivants : $INSERT$, $UPDATE$ et $DELETE$. Les anciennes et nouvelles valeurs sont spécifiées grâce à : OLD et : NEW.

```
CREATE [OR REPLACE] TRIGGER <nom declencheur>
BEFORE | AFTER
INSERT OR UPDATE [of <colonnes>] OR DELETE ON <table>
FOR EACH ROW
WHEN (<conditions>)
<instructions>
```

Exemple :

```
CREATE OR REPLACE TRIGGER MYTRIG
AFTER DELETE OR INSERT OR UPDATE ON BOOK
FOR EACH ROW
BEGIN
   IF DELETING THEN
      INSERT INTO XBOOK (PREVISBN, TITLE, DELDATE) VALUES (:OLD.ISBN, :OLD.TITLE, SYSDATE);
   ELSIF INSERTING THEN
      INSERT INTO NBOOK ......
   ELSIF UPDATING ('ISBN) THEN
      INSERT INTO CBOOK ......
   ELSE /* UPDATE TO ANYTHING ELSE THAN ISBN */
      INSERT INTO UBOOK ......
   END IF
END;
```

La commande **SHOW ERRORS** permet de visualiser les erreurs en cours.