PL/SQL

HLIN605

Pascal Poncelet
LIRMM
Pascal.Poncelet@lirmm.fr
http://www.lirmm.fr/~poncelet

Présentation

- PL/SQL : Programming Language with SQL
- Langage de programmation procédurale
- Langage Propriétaire Oracle mais qui ressemble beaucoup à de l'ADA

- Spécifiquement adapté à la manipulation de bases de données : types requêtes, curseurs, traitement des exceptions
- Permet de passer d'un monde ensembliste à un monde enregistrement par enregistrement

Présentation

- Côté serveur
  - Offre la possibilité de définir des objets persistants : procédures, fonctions, triggers
- Côté client
  - Permet d'écrire des blocs PL/SQL anonymes
  - Utilisable pour le développement d'interfaces graphiques, de masques de saisie (SQLForms), etc

Eléments de syntaxe

- Comme SQL la casse n'est pas importante
- Les identificateurs peuvent comporter des lettres, des chiffres, les caractères #, $, _
  *lettre (lettre | chiffre | # | $ |_)**
- Commentaires
  -- sur une ligne
  /
  sur …
  plusieurs lignes
  **/*

4

Un bloc PL/SQL

[**DECLARE**
Liste déclarations de variables, constantes, curseurs, exceptions]
[**BEGIN**]
Liste des instructions – Corps du bloc PL/SQL
[**EXCEPTION**
Gestion des exceptions]
[**END**] ;
/ -> le / indique exécution du bloc PL/SQL

5

Variables et constantes

- Les variables peuvent être de types suivants :
  – Scalaire, recevant une valeur de type SQL (CHAR, NUMBER, VARCHAR, …) ou de type PL/SQL (sous type prédéfini : INTEGER ou défini par l'utilisateur)
  – Composé (RECORD, collection, types objets)
  – Référence (REF) ou LOB (pour les données de grandes tailles)
- Les contraintes **NOT NULL** doivent être suivies d'une clause d'initialisation

identificateur [**CONSTANT**] typeDeDonnée [**NOT NULL**] [(:= | **DEFAULT**) expression];

6

Exemple de variables et constantes

```
var_emp_id NUMBER(6) = 8207;
dept VARCHAR2(10) NOT NULL := 'INFORMATIQUE';
effectif_max CONSTANT NUMBER(2,0) :=40 ;
disponible BOOLEAN := FALSE;
un_nombre NUMBER(5);
```

- Les déclarations multiples ne sont pas autorisés :

```
nom, prenom VARCHAR2(10); -- Interdit
```

- Les affectations des variables dans le bloc **BEGIN** … **END** suivent la syntaxe classique :=

```
un_nombre:=6;
```

7

Exemple de variables et constantes

```
DECLARE
    l_string VARCHAR2(20);
    l_number NUMBER(10);
    l_con_string CONSTANT VARCHAR2(20) := 'Ceci est une constante. ';
BEGIN
    l_string := 'Variable';
    l_number := 1;
    l_con_string := 'va échouer';
END;
/ -- exécution du bloc PL/SQL
l_con_string := 'va échouer'
*
ERROR at line 10:
ORA-06550: line 10, column 3:
PLS-00363: expression 'L_CON_STRING' cannot be used as an assignment target
ORA-06550: line 10, column 3:
PL/SQL: Statement ignored
SQL>
```

8

Les types composés

- Record : semblable à une structure C. Tous les types sont de type SQL. Une variable de type record peut ressembler à une ligne dans une relation
- Type collections : TABLE, VARRAY (relationnel-objet)
- Type Objets : relationnel-objet

```
TYPE PERSONNE IS RECORD (
Nom VARCHAR2(10),
Prenom VARCHAR2(10)
);
p PERSONNE_REC ;-- Accès possible aux champs via p.Nom et p.Prenom
```

9

Les types implicites

- Sont déclarés par :
  attribut**%TYPE**
- Signifie « du même type que »

numero PILOTE.Plnum**%TYPE** ;
– numero est du même type que l'attribut Plnum de la relation PILOTE
un_nombre **NUMBER**(4);
le_nombre un_nombre**%TYPE**;
– le_nombre est du même type qu'un_nombre

10

Les types implicites

- Sont déclarés par :
  attribut**%ROWTYPE**
- Signifie « du même type d'enregistrement que »

un_pilote PILOTE**%ROWTYPE** ;
– un_pilote est du même type que la relation pilote
– c'est à dire (Plnum, Plnom, Adr, Sal)
– on peut accéder à ses attributs par un_pilote.Plnum

11

Notion de sous type

- Chaque type prédéfini possède ses caractéristiques (domaine, opérateurs)
- Un sous type permet de restreindre les caractéristiques
- Il en existe des prédéfinis : **INTEGER, CHARACTER, POSITIVE**
- Il est possible de créer ses propres sous types
  SUBTYPE nom_sous_type **IS** type_de_base [(contraintes)] [**NOT NULL**];

SUBTYPE date_naissance_type **IS DATE NOT NULL**;
SUBTYPE les_categories **IS** PROF.CATEGORIE**%TYPE**;

12

Expressions et opérateurs

- Les opérateurs de SQL sont valides en PL/SQL
- Une opérande est une variable, une constante, un littéral, ou un appel à une fonction
- Opérateurs classiques :
  ** (exponentiation), +, -, *, /, <, >, =, <=, >=, <>, !=

  IS NULL, **LIKE**, **BETWEEN**, **IN**

  NOT, **AND**, **OR**

  +, -, || (opérateurs de concaténation de chaînes)

13

Instructions conditionnelles

- les instructions conditionnelles ont une syntaxe classique, comparable à celle d'ADA :

  IF <condition> **THEN [BEGIN]** <instructions> **[END]**
  [ELSIF <condition> **THEN [BEGIN]** <instructions> **[END]]**
  [ELSE [BEGIN] <instructions> **[END]]**
  END IF ;

14

Exemple

- Ce bloc augmente le salaire de l'employé 120 d'un bonus en fonction du nombre de ventes effectuées. La base de données est mise à jour.

```
DECLARE
 ventes  NUMBER(8,2)  := 12100;
 quota  NUMBER(8,2)  := 10000;
 bonus  NUMBER(6,2);
 emp_id  NUMBER(6)  := 120;
BEGIN
 IF  ventes > (quota+ 200) THEN
   bonus := (ventes  - quota)/4;
 ELSE
   bonus := 50;
 END  IF;
 UPDATE  EMP  SET  salaire  = salaire  + bonus WHERE  employe_id  = emp_id;
END;
/
```

15

Exemple

- Ce bloc augmente le salaire de l'employé 120 en fonction de sa catégorie (jobid)

```
DECLARE
  jobid       EMP.job_id%TYPE;
  empid       EMP.employe_id%TYPE  := 120;
  sal_augmentation   NUMBER(3,2);
BEGIN
  SELECT job_id INTO jobid  FROM EMP WHERE employe_id =
    empid;
  IF jobid = 'PROFESSEUR' THEN sal_augmentation  := .09;
  ELSIF jobid = 'MAITRE CONFERENCE' THEN sal_augmentation  :=
    .08;
  ELSIF jobid = 'ATER' THEN sal_augmentation := .07;
  ELSE sal_augmentation := 0;
  END IF;
UPDATE EMP SET ...
END;
/
```

16

Les itérations

- les instructions d'itération sont tout à fait classiques :
- Boucle **FOR** :

```
FOR <compteur> IN <borne_inf> .. <borne_sup>
LOOP
    <liste_instructions>
END LOOP ;
  /* il est inutile de déclarer <compteur> */
```

- Boucle **WHILE** :

```
WHILE
    <condition>
LOOP
    <liste_instructions>
END LOOP ;
```

17

Les itérations

- Il existe aussi la possibilité de sortir avec une clause **EXIT WHEN**
- Boucle **LOOP** :

```
LOOP
    <liste_instructions>
    EXIT WHEN <condition>
END LOOP ;
```

18

Exemple

```
BEGIN
   FOR num IN 0..10
LOOP
      DBMS_OUTPUT.put_line(TO_CHAR(num)) ;
END LOOP ;
END ;
/
```

- Affiche les 10 premiers nombres à l'écran
- Remarque : num n'a pas été déclaré dans les types utilisés

19

Exemple

```
DECLARE
   NUM NUMBER(2) := 0
BEGIN
LOOP
   INSERT INTO RESULTAT VALUES (NUM)
   NUM := NUM+1 ;
   EXIT WHEN  NUM > 10 ;
END LOOP ;
END ;
/
```

- Insère dans la relation RESULTAT les 10 premières valeurs de 0 à 9

20

Les branchements

- Le **EXIT WHEN** condition
- Le **GOTO :**

  GOTO <étiquette> ;

  – où <étiquette> est spécifiée dans le bloc

  – sous la forme : **<<** étiquette **>>**

21

Exemple

```
DECLARE
  p      VARCHAR(30);
  n      PLS_INTEGER := 37;
BEGIN
  FOR j in 2..ROUND(SQRT(n)) LOOP
   IF n MOD j = 0 THEN -- test nombre premier
     p := 'n"est  pas un nombre premier'; -- pas un nombre premier
     GOTO affiche_maintenant;
   END IF;
  END LOOP;
  p := ' est un nombre premier';
<<affiche_maintenant>>
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
/
```

22

Exploitation des requêtes SQL

- Les instructions **SELECT**, **INSERT**, **DELETE**, **UPDATE** peuvent être utilisées dans un bloc
- Elles peuvent utiliser des variables du programme mais attention les types doivent être compatibles et il faut utiliser des noms de variables différents
- Il est possible d'affecter le retour d'une requête qui contient une seule valeur dans une variable avec **SELECT** ... **INTO**
- Pour les requêtes qui retournent plusieurs tuples il faut utiliser les curseurs (voir plus loin)

23

Exemple

```
DECLARE
   RESTE NUMBER := 7324 ;
BEGIN
WHILE RESTE >=9 LOOP
   RESTE := RESTE-9 ;
END LOOP ;
INSERT INTO RESULTAT VALUES (reste, 'reste division 7324 par 9');
END ;
/
```

- Sauvegarde dans la relation RESULTAT le contenu de la division de 7324 par 9

24

Exemple

- Récupération du nombre de vols stockés dans la base :

```
DECLARE
nb_vol   NUMBER(4,0) ;
...
BEGIN
SELECT COUNT(*) INTO nb_vol FROM VOL;
...
END ;
```

- Récupération d'un tuple de la relation VOL de la base :

```
DECLARE
un_vol   VOL%ROWTYPE;
...
BEGIN
SELECT * INTO un_vol FROM VOL
WHERE Volnum='AF523';
...
END ;
```

25

Les curseurs

- Rôle : établir la transition entre l'univers BD et celui des langages procéduraux classiques
- Permettent de pouvoir manipuler un à un tous les tuples retournés par une requête

- Un curseur est défini dans la partie déclarative d'un bloc PL/SQL par une requête d'interrogation en SQL (sa structure correspond aux attributs du SELECT), en suivant la syntaxe suivante :

CURSOR <nom_curseur> **IS** <requête_SQL> **;**

26

Exemple

```
DECLARE
    une_variable NUMBER(4);
  CURSOR C_pilote IS
    SELECT Plnum, Plnom
    FROM PILOTE
    ORDER BY Plnum, Plnom;
BEGIN
...
END;
/
```

27

Gestion des curseurs

- Dans le corps du bloc entre **BEGIN** et **END**

OPEN <nom_curseur> ;
exécute la requête de définition du curseur et alloue la place mémoire nécessaire. Le curseur peut alors être perçu comme une suite d'enregistrements.

CLOSE <nom_curseur> ;
désactive le curseur et libère la place mémoire. Le curseur est alors perçu comme un ensemble indéfini.

FETCH <nom_curseur> **INTO** <liste_variables> ;
ramène le prochain enregistrement du curseur et renseigne les différentes variables réceptrices.

28

Attributs des curseurs

- Il s'agit de propriétés booléennes prédéfinies des curseurs

<nom_curseur>**%NOTFOUND**
est à vrai si l'ordre FETCH ne retourne aucun enregistrement.

<nom_curseur>**%FOUND**
est à vrai si l'ordre FETCH retourne un enregistrement.

<nom_curseur>**%ISOPEN**
est à vrai si le curseur est ouvert

<nom_curseur>**%ROWCOUNT**
retourne le nombre de tuples qui ont été accédés via le curseur (0 avant le 1er **fetch**, puis 1, puis 2 …).

29

Exemple

```
DECLARE
CURSOR MesPilotesParisiens IS
   SELECT * FROM PILOTE
   WHERE Adr= 'PARIS';
mon_pilote pilote%ROWTYPE;
BEGIN
   OPEN MesPilotesParisiens;
   LOOP
      FETCH MesPilotesParisiens INTO mon_pilote;
      DBMS_OUTPUT.PUT_LINE(mon_pilote.Plnom);
      EXIT WHEN MesPilotesParisiens%NOTFOUND;
   END LOOP;
   CLOSE MesPilotesParisiens;
END;
```

30

Exemple

- %ISOPEN

```
IF NOT lecurseur%ISOPEN THEN
    OPEN lecurseur;
END IF;
```

- %FOUND

```
OPEN lecurseur;
LOOP
  FETCH lecurseur INTO variable1,variable2;
  EXIT WHEN NOT lecurseur%FOUND;
END LOOP;
CLOSE lecurseur;
```

31

Exemple

- %NOTFOUND

```
OPEN lecurseur;
LOOP
  FETCH lecurseur INTO variable1,variable2;
  EXIT WHEN lecurseur%NOTFOUND;
END LOOP;
CLOSE lecurseur;
```

- Dans un while

```
OPEN lecurseur;
  FETCH lecurseur INTO variable1,variable2;
  WHILE lecurseur%FOUND
        LOOP
            ...
            FETCH lecurseur INTO variable1,variable2;
        END LOOP;
CLOSE lecurseur;
```

32

Les curseurs

- Si l'ordre **SELECT** de définition du curseur comporte un calcul (horizontal ou vertical), il faut attribuer un alias au calcul pour pouvoir le manipuler ultérieurement.

```
CURSOR comptage IS
SELECT VD, COUNT(*) nb_arrivees
FROM VOL
GROUP BY VD;
```

- Le nombre de vols desservant chaque ville peut alors être manipulé par comptage.nb_arrivees

33

Les curseurs

- Il est possible de laisser le système gérer les curseurs sans utiliser **OPEN**, **FETCH**, **CLOSE** ni de déclaration de variable

```
FOR <nom_variable> IN <nom_curseur >
LOOP <liste_instructions> END LOOP;
```

```
DECLARE
  CURSOR comptage IS
  SELECT VD, COUNT(*) nb_arrivees
  FROM VOL
  GROUP BY VD;
BEGIN
  FOR C1 IN comptage LOOP
        IF C1.nb_arrivees < 10 THEN …
  END LOOP;
END
```

Les curseurs

- Il est possible de passer un paramètre à un curseur

```
DECLARE
  CURSOR lecurseur (un_car CHARACTER) IS SELECT att1,att2
  FROM table
  WHERE att3 = un_car;
BEGIN
  OPEN lecurseur('a');
```

Les exceptions

- Rappel : les exceptions permettent de contrôler des erreurs d'exécution. Affichage de l'erreur ou traitement de l'erreur.
- Il existe deux types d'exception :
  - Exceptions définies par l'utilisateur dans la partie déclarative du bloc. Elles sont déclenchées dans le corps du bloc, si une condition est remplie, par :

```
IF <condition> THEN RAISE <nom_exception>; END IF ;
```

  - Exceptions prédéfinies, gérées par ORACLE, correspondant à des erreurs internes.

Les exceptions

- *Quelques exemples* :
  - **NO_DATA_FOUND** : déclenchée si une requête ne rend aucun résultat ;
  - **ZERO_DIVIDE** : déclenchée s'il y a tentative de division par 0 ;
  - **DUP_VAL_ON_INDEX** : déclenchée lors d'une tentative d'insertion d'une valeur dupliquée pour un attribut sur lequel est défini un index primaire ;
  - **INVALID_NUMBER** : déclenchée si une incompatibilité pour un type numérique est détectée.
  - **INVALID_CURSOR** déclenchée par exemple dans le cas d'accès à un curseur non ouvert.

37

Les exceptions

- Le traitement des exceptions se fait dans la partie **EXCEPTION** du bloc PL/SQL par :

WHEN <nom_exception> **THEN** [**BEGIN**] <liste_instructions> [**END**] **;**

Ou

WHEN OTHERS THEN <liste_instructions>

38

Exemple

```
DECLARE
   nb_vols NUMBER(2,0) ;
   impossible EXCEPTION ;
   numero VOL.Volnum%TYPE ;
   …
BEGIN
   SELECT COUNT(*) INTO nb_vols FROM VOL;
   …
   IF numero > 10000 THEN RAISE impossible ;
   …
EXCEPTION
   WHEN impossible THEN numero := 0 ;
   WHEN OTHERS THEN numero := 100 ;
END ;
```

39

Exemple

```
DECLARE
   ratio NUMBER(3,1);
BEGIN
   SELECT valeur / nombre INTO ratio FROM table;
   -- peut entraîner une division par 0
   INSERT INTO STATS (chaine,ratio) VALUES ('la valeur est', ratio);
EXCEPTION
-- Traitement de l'exception
WHEN ZERO_DIVIDE THEN INSERT INTO STATS (ratio) VALUES ('Division par
   0', NULL);
END;
```

40

Exemple

```
DECLARE
  valeur INTEGER := 7;
BEGIN
  IF valeur NOT IN (1, 2, 3) THEN RAISE INVALID_NUMBER;
  END IF;
EXCEPTION
  WHEN INVALID_NUMBER THEN ROLLBACK;
  WHEN OTHERS THEN .....
END;
```

41

Les modules stockés

- Un module stocké est un programme rangé dans la base de données et être ainsi ré-utilisables et partageables (autorisation)
- Ces programmes peuvent être appelés à tout moment par un client et seront exécutés sur le serveur
- Il est possible de définir des procédures ou des fonctions

42

Les procédures

```
CREATE[OR REPLACE] PROCEDURE nom_procedure
/* Déclaration des paramètres */
    (var_entree IN type,
    var_sortie OUT type,
    var_entrée_sortie IN OUT type) IS
/* Déclaration des variables locales*/
    var_locale type;
BEGIN
  <liste_instructions>
[EXCEPTION ...]
END ;
```

43

Les fonctions

```
CREATE [OR REPLACE] FUNCTION nom_fonction
/* Déclaration des paramètres */
 (var_entree IN type, ...)
RETURN type IS
  /* Déclaration des variables locales*/
  var_locale type;
BEGIN
  <liste_instructions>
  RETURN (var_locale) ;
  [EXCEPTION ...]
END ;
```

44

Exemple

```
CREATE OR REPLACE FUNCTION nb_vol ( num IN INTEGER)
RETURN INTEGER IS
  nb INTEGER ;
BEGIN
  SELECT COUNT(Volnum) INTO nb
  FROM VOL
  WHERE Plnum= num ;
  RETURN (nb) ;
END ;
```

45

Exemple

```
CREATE OR REPLACE PROCEDURE nom_pil (
  numero IN PILOTE.Plnum%type,
  nom OUT PILOTE.Plnom%type) IS
BEGIN
  SELECT Plnom INTO nom FROM PILOTE WHERE numero = Plnum ;
END;
```

- Appel

```
DECLARE
  LeNomPilote VARCHAR(100) ;
BEGIN
    nom_pil(100,leNomPilote) ;-- appel de la procedure
    dbms_output.put_line(leNomPilote) ; -- affichage
END;
```

46

Entrées-Sorties

- Pour pouvoir **afficher du texte à l'écran** utilisation du package DBMS_OUTPUT : au niveau du prompt SQLPlus, exécuter l'instruction suivante :

  SET SERVEUROUTPUT ON

- l'instruction pour afficher du texte ou le contenu d'une variable est :

  DBMS_OUTPUT.PUT_LINE('texte' || variable) ;
- où || est le caractère permettant la concaténation de chaînes.

47

Entrées-Sorties

- D'autres fonctions disponibles pour l'affichage

dbms_output.enable (autorise l'affichage)

dbms_output.disable (interdit l'affichage)

dbms_output.put_line (affiche la chaine et passe à la ligne)

dbms_output.new_line (passe à la ligne)

48

Entrées-Sorties

- Dans votre code :

```
SET SERVEROUTPUT ON;
DECLARE
  LeNomPilote VARCHAR(100) ;
BEGIN
    nom_pil(100,leNomPilote)   ; -- appel de la procedure
    dbms_output.put_line( leNomPilote)   ; -- affichage
END;
```

- Des questions ?