SGBD - 2e

PL-SQL - Chapitre 6 - Des records aux collections bulk

Daniel Schreurs

8 février 2022

Haute École de Province de Liège

Table des matières du chapitre i

Introduction

Table des matières de la section : Introduction i

Introduction : Définitions

Important

Les records [1] permettent de définir des structures de données hétérogènes.

1. Coir structures en langage C

Introduction : Exemple 1

Exemple 1

```
1  CREATE TABLE Dept
2  (
3      DeptNo NUMBER(2)
4          CONSTRAINT CPDept PRIMARY KEY,
5      Dname  VARCHAR2(14),
6      Loc    VARCHAR2(13)
7  );
8
9  declare
10     TYPE TypeDept IS RECORD
11                      (
12                          DeptNo NUMBER,
13                          Dname  VARCHAR2(14),
14                          Loc    VARCHAR2(13)
15                      );
16     UnDepartement TypeDept;
17 BEGIN
```

Introduction : Exemple 1

Exemple 1 - avec %type

```
declare
    TYPE TypeDept IS RECORD
                     (
                         DeptNo Dept.DeptNo%TYPE,-- Type
                         Dname  Dept.Danme%TYPE,
                         Loc    Dept.Loc%TYPE
                     );
    UnDepartement TypeDept;
BEGIN
    null;
END;
```

Introduction : Exemple 1

Exemple 1 - avec %ROWTYPE

```
1 declare
2     UnDepartement Dept%ROWTYPE;
3 BEGIN
4     null;
5 END;
```

Peut-on se passer des types record ?

Opérateurs

Table des matières de la section : Opérateurs i

Table des matières de la section : Opérateurs ii

Opérateurs : Opérateur d'accession

Exemple 2

```
DECLARE
  UnDepartement  Dept%ROWTYPE;
  VDeptno        Dept.Deptno%TYPE;
BEGIN
    -- C'est avec le "." qu'on accède à la propriété
  UnDepartement.Deptno := 99;
  UnDepartement.Dname := 'Inpres';
  VDeptno := UnDepartement.Deptno;

  DBMS_OUTPUT.PUT_LINE ('Nom département : ' ||
            UnDepartement.Dname);
END;
```

Opérateurs : Comparaison

Important

Pour **comparer deux records**, il faut les comparer champ par champ. L'idéal est donc d'écrire une fonction qui prend 2 records en paramètres et renvoie vrai ou faux

Opérateurs : Comparaison

Comparaison erronée

```
1  DECLARE
2      UnDepartement  Dept%ROWTYPE;
3      UnDepartement2 Dept%ROWTYPE;
4  BEGIN
5      -- ERREUR
6      IF UnDepartement = UnDepartement2 -- ORA-06550
7      THEN
8          DBMS_OUTPUT.PUT_LINE('OK');
9      END IF;
10 END;
```

Opérateurs : Comparaison

Comparaison dangereuse

```
1  DECLARE
2      UnDepartement  Dept%ROWTYPE;
3      UnDepartement2 Dept%ROWTYPE;
4  BEGIN
5      IF UnDepartement.Deptno = UnDepartement2.Deptno
6          AND UnDepartement.Dname = UnDepartement.Dname
7          AND UnDepartement.Loc = UnDepartement.Loc
8      THEN
9          DBMS_OUTPUT.PUT_LINE('OK');
10     ELSE
11         DBMS_OUTPUT.PUT_LINE('NOK');
12     END IF;
13 END ;
```

Opérateurs : Comparaison

Comparaison OK

```
DECLARE
    UnDepartement  Dept%ROWTYPE;
    UnDepartement2 Dept%ROWTYPE;
BEGIN
    IF COALESCE(UnDepartement.Deptno, 0) =
       COALESCE(UnDepartement2.Deptno, 0)
        AND COALESCE(UnDepartement.Dname, 'X') =
            COALESCE(UnDepartement.Dname, 'X')
        AND COALESCE(UnDepartement.Loc, 'X') =
            COALESCE(UnDepartement.Loc, 'X')
    THEN
        DBMS_OUTPUT.PUT_LINE('OK');
    ELSE
        DBMS_OUTPUT.PUT_LINE('NOK');
    END IF;
END;
```

Opérateurs : Affectation multiple

INSERT INTO

```
DECLARE
    UnDepartement Dept%ROWTYPE;
BEGIN
    UnDepartement.Deptno := 9;
    UnDepartement.Dname := 'Inpres';
    UnDepartement.Loc := 'Seraing';
    INSERT INTO Dept VALUES UnDepartement;
    COMMIT;
END;
```

Opérateurs : Affectation multiple

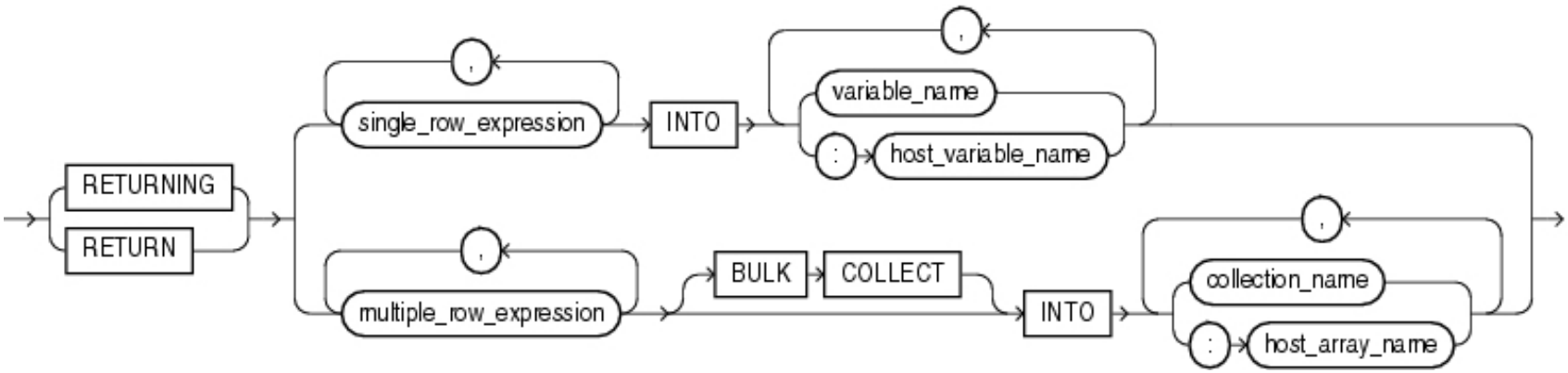
UPDATE Dept SET

```
1  DECLARE
2      UnDepartement Dept%ROWTYPE;
3  BEGIN
4      UnDepartement.Deptno := 9;
5      UnDepartement.Dname := 'Rennequins';
6      UnDepartement.Loc := 'Liège';
7
8      UPDATE Dept SET ROW = UnDepartement WHERE Deptno =
           9;
9      IF SQL%NOTFOUND
10     THEN
11         DBMS_OUTPUT.PUT_LINE('Pas de MAJ');
12     ELSE
13         DBMS_OUTPUT.PUT_LINE('Mise à jour OK');
14     END IF;
15     COMMIT;
16 END;
```

Opérateurs : RETURNING INTO

Important

L'utilisation de la clause `RETURNING INTO` dans les commandes `insert`, `update` et `delete` évite de refaire une nouvelle lecture pour connaitre les nouvelles (ou anciennes) valeurs d'une ligne !



Opérateurs : RETURNING INTO

UPDATE Dept SET

```
1  DECLARE
2      UnEmploye Employe%ROWTYPE;
3  BEGIN
4      UPDATE emp
5      SET sal = sal * 1.1
6      WHERE job = 'PRESIDENT'
7      RETURNING Ename, Sal INTO UnEmploye; -- ici
8      IF SQL%FOUND
9      THEN
10         DBMS_OUTPUT.PUT_LINE
11             (UnEmploye.Ename || ' ' || UnEmploye.Sal);
12     ELSE
13         DBMS_OUTPUT.PUT_LINE('Pas Maj');
14     END IF;
15     COMMIT;
16 END;
```

Opérateurs : RETURNING INTO

Rappel ! Recherche d'UN tuple

```
1 DECLARE
2     UnEmploye Emp%ROWTYPE;
3 BEGIN
4     SELECT * INTO UnEmploye FROM Emp WHERE Empno = 7902;
5     DBMS_OUTPUT.PUT_LINE(UnEmploye.Ename);
6 EXCEPTION
7     WHEN NO_DATA_FOUND THEN
8         DBMS_OUTPUT.PUT_LINE('Pas d''employé');
9 END;
```

2

---

2. Attention, si critère de recherche ne porte pas sur la clé primaire, prévoir également le TOO_MANY_ROWS !!!

Opérateurs : BULK COLLECT INTO

Recherche de PLUSIEURS tuples

```
1  DECLARE
2      TYPE TableEmployes IS TABLE OF Emp%ROWTYPE
3          INDEX BY BINARY_INTEGER;
4      LesEmployes TableEmployes;
5  BEGIN
6      -- Ici BULK COLLECT INTO
7      SELECT * BULK COLLECT INTO LesEmployes FROM Emp
           WHERE Deptno = 10;
8      IF LesEmployes.FIRST IS NOT NULL THEN
9          FOR i IN LesEmployes.FIRST..LesEmployes.LAST
10             LOOP
11                 DBMS_OUTPUT.PUT_LINE(LesEmployes(i).
                       Ename);
12             END LOOP;
13     END IF;
14 END;
```

Opérateurs : BULK COLLECT INTO

Important

Avec `BULK COLLECT INTO`, `NO_DATA_FOUND` n'est pas déclenchée lorsque le résultat de la recherche est vide !

Opérateurs : FORALL

- L'instruction `FORALL` permet d'envoyer en une fois un lot d'instructions SQL et les collections `bulk` permettent de récupérer les résultats ;
- Pour accélérer le traitement des instructions insert, update ou delete, on les place dans un forall plutôt que dans une boucle classique. Pour optimiser les instructions select, on utilise la clause into bulk collect !

Opérateurs : FORALL

Rappel : la boucle FOR

```
1  DECLARE
2      TYPE TableEmployes IS TABLE OF Emp%ROWTYPE
3          INDEX BY BINARY_INTEGER;
4      LesEmployes TableEmployes;
5      Nbre         NUMBER := 1;
6  BEGIN
7      FOR UnEmploye IN (SELECT * FROM Emp)
8          LOOP
9              LesEmployes(Nbre) := UnEmploye;
10             Nbre := Nbre + 1;
11         END LOOP;
12 END;
```

Opérateurs : FORALL i

FORALL i IN

```
1  CREATE TABLE empTemp AS
2  SELECT *
3  FROM emp;
4  -- les éléments de la collection possèdent des indices
       consécutifs
5  DECLARE
6      TYPE TableNrEmployes IS TABLE OF Emp.Empno%TYPE
7          INDEX BY BINARY_INTEGER;
8      LesNrEmployes TableNrEmployes;
9  BEGIN
10     LesNrEmployes(1) := 1234;
11     LesNrEmployes(2) := 9999; -- employé 9999 'nexiste
           pas
12     LesNrEmployes(3) := 7934;
```

Opérateurs : FORALL ii

```
13      FORALL i IN LesNrEmployes.FIRST..LesNrEmployes.LAST
14          DELETE FROM empTemp WHERE empno = LesNrEmployes(
                i);
15      COMMIT;
16  END;
```

Opérateurs : FORALL i

Récupérer les employés supprimés

```
1  DECLARE
2      TYPE TableNrEmployes IS TABLE OF Emp.Empno%TYPE
3          INDEX BY BINARY_INTEGER;
4      LesNrEmployes TableNrEmployes;
5      TYPE TableEmployes IS TABLE OF Emp%ROWTYPE
6          INDEX BY BINARY_INTEGER;
7      LesEmployes   TableEmployes;
8  BEGIN
9      LesNrEmployes(1) := 1234;
10     LesNrEmployes(2) := 9999; -- employé 9999 'nexiste
           pas
11     LesNrEmployes(3) := 7934;
12     FORALL i IN LesNrEmployes.FIRST..LesNrEmployes.LAST
13         DELETE
```

Opérateurs : FORALL ii

```
14          FROM empTemp
15          WHERE empno = LesNrEmployes(i)
16          RETURNING empno,ename,job,mgr,hiredate,sal,comm,
                deptno BULK COLLECT INTO LesEmployes;
17      DBMS_OUTPUT.PUT_LINE('Deleted ' || SQL%ROWCOUNT || '
            tuple(s)');
18      COMMIT;
19  end;
```

Opérateurs : FORALL i

Important

Dans une instruction `FORALL`, si l'exécution d'une instruction SQL provoque le déclenchement d'une **exception** non gérée, toutes les modifications réalisées par **les instructions précédentes seront annulées** [3].

3. Sauf si l'exception est gérée.

Opérateurs : FORALL i

FORALL et exception gérée

```
1  DECLARE
2      TYPE TableNrDept IS TABLE OF Dept.Deptno%TYPE
3          INDEX BY BINARY_INTEGER;
4      LesNrDept TableNrDept;
5  BEGIN
6      LesNrDept(1) := 50;
7      LesNrDept(2) := 40; -- département 40 existe
8      LesNrDept(3) := 60;
9      FORALL i IN LesNrDept.FIRST..LesNrDept.LAST
10         INSERT INTO Dept(deptno, dname)
11         VALUES (LesNrDept(i), 'Dept' || LesNrDept(i));
12     COMMIT;
13 EXCEPTION
14     WHEN OTHERS THEN -- Gestion médiocre...
```

Opérateurs : FORALL ii

```
15          COMMIT;
16  END ;
```

Opérateurs : SQL%BULK_ROWCOUNT

Le nombre de lignes impliquées dans chaque instruction LMD d'un FORALL peut être déterminé au moyen de l'attribut composé `SQL%BULK_ROWCOUNT`.[4]

4. Cet attribut est un tableau PL/SQL dont le nième élément contient le nombre de lignes traitées dans la nième instruction LMD contenue dans un FORALL

Opérateurs : save exceptions

- PL/SQL possède un mécanisme pour gérer les exceptions déclenchées pendant l'exécution d'un FORALL.
- Ce mécanisme, déclenché par la clause `save exceptions`, permet de sauver les informations sur les exceptions et de continuer le traitement des instructions.
- L'absence de cette clause dans une instructions FORALL provoque l'arrêt de l'instruction dès la première exception rencontrée.

Opérateurs : save exceptions

Avec la clause SAVE EXCEPTIONS, toutes les exceptions détectées pendant l'exécution de FORALL sont sauvées dans l'attribut `%BULK_EXCEPTIONS (table PL_SQL)`.

- `SQL%BULK_EXCEPTIONS.COUNT` : nombre d'exceptions rencontrées pendant l'exécution du FORALL
- `SQL%BULK_EXCEPTIONS(i).ERROR_INDEX` : indice de l'itération qui a provoqué l'exception
- `SQL_BULK_EXCEPTIONS(i).ERROR_CODE` : code d'erreur d'Oracle

Opérateurs : save exceptions i

Gestion des exceptions dans FORALL

```
1  DECLARE
2      TYPE TableNrDept IS TABLE OF Dept.Deptno%TYPE INDEX
           BY BINARY_INTEGER;
3      LesNrDept TableNrDept;
4  BEGIN
5      LesNrDept(1) := 50;
6      LesNrDept(2) := 40; -- département 40 existe
7      LesNrDept(3) := 60;
8      FORALL i IN LesNrDept.FIRST..LesNrDept.LAST SAVE
           EXCEPTIONS
9          INSERT INTO Dept (deptno, dname) VALUES (
               LesNrDept(i), 'Dept' || LesNrDept(i));
10     COMMIT;
11 EXCEPTION
```

Opérateurs : save exceptions ii

```
12      WHEN OTHERS THEN COMMIT;
13      FOR i IN 1 .. SQL%BULK_EXCEPTIONS.COUNT
14          LOOP
15              DBMS_OUTPUT.PUT_LINE('Erreur pour le nr de
                    dépt ' ||
16                                  LesNrDept(SQL%
                                        BULK_EXCEPTIONS(i).
                                        ERROR_INDEX) || ' '
                                        ||
17                                  SQLERRM(-SQL%
                                        BULK_EXCEPTIONS(i).
                                        ERROR_CODE));
18          END LOOP;
19  end;
```

Opérateurs : save exceptions i

RETURNING...BULK COLLECT INTO

```
1  DECLARE
2      TYPE TableNrDept IS TABLE OF Dept.Deptno%TYPE INDEX
           BY BINARY_INTEGER;
3      LesNrDept TableNrDept;
4  BEGIN
5      LesNrDept(1) := 50;
6      LesNrDept(2) := 40; -- département 40 existe
7      LesNrDept(3) := 60;
8      FORALL i IN LesNrDept.FIRST..LesNrDept.LAST SAVE
           EXCEPTIONS
9          INSERT INTO Dept (deptno, dname) VALUES (
               LesNrDept(i), 'Dept' || LesNrDept(i));
10     COMMIT;
11 EXCEPTION
```

Opérateurs : save exceptions ii

```
12      WHEN OTHERS THEN COMMIT;
13      FOR i IN 1 .. SQL%BULK_EXCEPTIONS.COUNT
14          LOOP
15              DBMS_OUTPUT.PUT_LINE('Erreur pour le nr de
                    dépt ' ||
16                                  LesNrDept(SQL%
                                        BULK_EXCEPTIONS(i).
                                        ERROR_INDEX) || ' '
                                        ||
17                                  SQLERRM(-SQL%
                                        BULK_EXCEPTIONS(i).
                                        ERROR_CODE));
18          END LOOP;
19  end;
```

Résumé

Table des matières de la section : Résumé i

Résumé : Recherche d'un seul tuple

```
SELECT ... INTO ... FROM ... WHERE ... ;
```

- Le tuple est trouve : l'exécution continue
- Le tuple n'est pas trouvé : exception `NO_DATA_FOUND`
- La sélection renvoie plus d'un tuple exception `TOO_MANY_ROWS`

Résumé : Recherche d'un seul tuple

La boucle FOR

```
1 FOR VariableImplicite
2   IN (SELECT ... FROM  ... WHERE ...)
3 LOOP
4   --Traitement d'un tuple
5 END LOOP;
```

- Un ou plusieurs tuples sélectionnés => exécution LOOP
- Aucun tuple sélectionné => pas exécution LOOP, on continue l'exécution après LOOP
- PAS d'exception `NO_DATA_FOUND` déclenchée !!!

Résumé : Recherche de plus d'un tuple, initialisation d'une collection de records

BULK COLLECT INTO

```
1 SELECT * BULK COLLECT INTO VariableTable
2 FROM ... WHERE ... ;
```

- Un ou plusieurs tuples sélectionnés => initialisation de la VariableTable indicée de 1 à n
- Aucun tuple sélectionné => on continue l'exécution après le `SELECT`
- PAS d'exception `NO_DATA_FOUND` déclenchée !!!

Résumé : BULK_EXCEPTIONS

- Option `SAVE EXCEPTIONS` permet de sauver toutes les exceptions détectées pendant l'exécution de `FORALL` dans une table PL/SQL et de continuer le traitement.
- `SQL%BULK_EXCEPTIONS.COUNT` le nombre d'exceptions rencontrées pendant l'exécution du `FORALL`
- `SQL%BULK_EXCEPTIONS(i).ERROR_INDEX` contient l'indice de l'itération qui a provoqué l'exception
- `SQL%BULK_EXCEPTIONS(i).ERROR_CODE` contient le code d'erreur d'Oracle
- Le nième élément de `SQL%BULK_ROWCOUNT` contient le nombre de lignes traitées dans la nième instructions LMD