

SGBD - 2^e

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

Daniel Schreurs

14 février 2022

Haute École de Province de Liège

Table des matières du chapitre i

1. Introduction

2. Opérateurs

3. Résumé

Introduction

Table des matières de la section : Introduction i

1. Introduction

1.1 Définitions

1.2 Exemple 1

2. Opérateurs

3. Résumé

Important

Les records¹ 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 declare
9     TYPE TypeDept IS RECORD
10         (
11             DeptNo NUMBER,
12             Dname  VARCHAR2(14),
13             Loc    VARCHAR2(13)
14         );
15     UnDepartement TypeDept;
16 BEGIN
17     null;
```

Introduction : Exemple 1

Exemple 1 - avec %type

```
1 declare
2     TYPE TypeDept IS RECORD
3         (
4             DeptNo Dept.DeptNo%TYPE, -- Type
5             Dname   Dept.Danme%TYPE,
6             Loc     Dept.Loc%TYPE
7         );
8     UnDepartement TypeDept;
9 BEGIN
10     null;
11 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

1. Introduction

2. Opérateurs

2.1 Opérateur d'accession

2.2 Comparaison

2.3 Affectation multiple

2.4 RETURNING INTO

2.5 BULK COLLECT INTO

2.6 FORALL

2.7 SQL%BULK_ROWCOUNT

2.8 save exceptions

3. Résumé

Opérateurs : Opérateur d'accesion

Exemple 2

```
1 DECLARE
2   UnDepartement  Dept%ROWTYPE;
3   VDeptno        Dept.Deptno%TYPE;
4 BEGIN
5   -- C'est avec le "." qu'on accède à la propriété
6   UnDepartement.Deptno := 99;
7   UnDepartement.Dname := 'Inpres';
8   VDeptno := UnDepartement.Deptno;
9   DBMS_OUTPUT.PUT_LINE ('Nom département : ' ||
10                          UnDepartement.Dname);
11 END;
```

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

```
1 DECLARE
2     UnDepartement Dept%ROWTYPE;
3     UnDepartement2 Dept%ROWTYPE;
4 BEGIN
5     IF COALESCE(UnDepartement.Deptno, 0) =
6         COALESCE(UnDepartement2.Deptno, 0)
7         AND COALESCE(UnDepartement.Dname, 'X') =
8             COALESCE(UnDepartement.Dname, 'X')
9         AND COALESCE(UnDepartement.Loc, 'X') =
10            COALESCE(UnDepartement.Loc, 'X')
11     THEN
12         DBMS_OUTPUT.PUT_LINE('OK');
13     ELSE
14         DBMS_OUTPUT.PUT_LINE('NOK');
15     END IF;
16 END;
```

Opérateurs : Affectation multiple

INSERT INTO

```
1 DECLARE
2     UnDepartement Dept%ROWTYPE;
3 BEGIN
4     UnDepartement.Deptno := 9;
5     UnDepartement.Dname := 'Inpres';
6     UnDepartement.Loc := 'Seraing';
7     INSERT INTO Dept VALUES UnDepartement;
8     COMMIT;
9 END;
```

Opérateurs : Affectation multiple

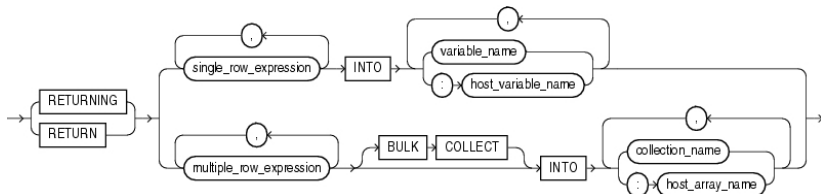
UPDATE Dept SET

```
1 DECLARE
2     UnDepartement Dept%ROWTYPE;
3 BEGIN
4     UnDepartement.Deptno := 9;
5     UnDepartement.Dname := 'Rennequins';
6     UnDepartement.Loc := 'Liège';
7     UPDATE Dept SET ROW = UnDepartement WHERE Deptno =
8         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 connaître les nouvelles (ou anciennes) valeurs d'une ligne !



Opérateurs : RETURNING INTO

UPDATE Dept SET

```
1 DECLARE
2     UnEmployee Employee%ROWTYPE;
3 BEGIN
4     UPDATE emp
5     SET sal = sal * 1.1
6     WHERE job = 'PRESIDENT'
7     RETURNING Ename, Sal INTO UnEmployee; -- ici
8     IF SQL%FOUND
9     THEN
10         DBMS_OUTPUT.PUT_LINE
11             (UnEmployee.Ename || ' ' || UnEmployee.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
8         WHERE Deptno = 10;
9     IF LesEmployes.FIRST IS NOT NULL THEN
10         FOR i IN LesEmployes.FIRST..LesEmployes.LAST
11             LOOP
12                 DBMS_OUTPUT.PUT_LINE(LesEmployes(i).
13                     Ename);
14             END LOOP;
15     END IF;
16 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 !

- 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 'n'existe
   pas
12    LesNrEmployes(3) := 7934;
```

```
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 'n'existe
    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;
```

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...
```



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

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

- 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 instruction FORALL provoque l'arrêt de l'instruction dès la première exception rencontrée.

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
3     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 SAVE
10     EXCEPTIONS
11         INSERT INTO Dept (deptno, dname) VALUES (
12             LesNrDept(i), 'Dept' || LesNrDept(i));
13 COMMIT;
14 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
3     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 SAVE
10     EXCEPTIONS
11     INSERT INTO Dept (deptno, dname) VALUES (
12         LesNrDept(i), 'Dept' || LesNrDept(i));
13 COMMIT;
14 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

1. Introduction

2. Opérateurs

3. Résumé

3.1 Recherche d'un seul tuple

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

3.3 BULK_EXCEPTIONS

Résumé : Recherche d'un seul tuple

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

- Le tuple est trouvé : 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 instruction LMD