# Axes d'améliorations PL/SQL

## Curseurs implicites

Il n’est plus nécessaire de déclarer un curseur dans la section déclarative ainsi d’appeler les commandes d’ouverture, de fetch et de fermeture.

Cela permet une meilleure lisibilité du code et moins d’oublis de fermeture de curseur.

|  |  |  |
| --- | --- | --- |
| procedure curseur\_explicite is  cursor cur is select \* from rp\_chevaliers;  rec cur%rowtype;  begin  open cur;  fetch cur into rec;  while cur%found  loop  null ;  fetch cur into rec;  end loop;  close cur;  exception  when others then  close cur;  raise;  end; | devient | procedure curseur\_implicite  is  begin  for rec in (select \* from rp\_chevaliers)  loop  -- traitement  null;  end loop;  end; |

## Commandes (limiter le nombre d'accès BdD)

### Forall

Permet de ne faire qu'un accès base pour plusieurs DML (insert, update, delete).

CREATE TABLE employe AS SELECT \* FROM emp;

DECLARE

TYPE NumList IS VARRAY(20) OF NUMBER;

depts NumList := NumList(10, 30, 70);

BEGIN

FORALL i IN depts.FIRST..depts.LAST

DELETE FROM employe WHERE deptno = depts(i);

--COMMIT;

END;

/

select \* from employe;

### Bulk collect

Permet de récupérer directement tout un resultSet dans une liste en une seule commande

#### Avec un select

-- Chargement d'un result set via bulk collect

DECLARE

type tab is table of emp%rowtype index by binary\_integer;

monTab tab;

nb pls\_integer;

begin

select \* bulk collect into monTab from emp;

nb := monTab.count;

dbms\_output.put\_line(nb);

end bulk\_in\_table;

/

Autre exemple

DECLARE

TYPE NumTab IS TABLE OF emp.empno%TYPE;

TYPE NameTab IS TABLE OF emp.ename%TYPE;

enums NumTab; -- No need to initialize the collections.

names NameTab; -- Values will be filled in by the SELECT INTO.

PROCEDURE print\_results IS

BEGIN

dbms\_output.put\_line('Results:');

FOR i IN enums.FIRST .. enums.LAST LOOP

dbms\_output.put\_line(' Employee #' || enums(i) || ': ' ||names(i));

END LOOP;

END;

BEGIN

SELECT empno, ename -- extraction de 10 employés

BULK COLLECT INTO enums, names FROM emp

WHERE ROWNUM < 11;

print\_results;

SELECT empno, ename -- extraction de 20% des employés

BULK COLLECT INTO enums, names FROM emp SAMPLE (20);

print\_results;

END;

/

#### Avec un curseur

DECLARE

TYPE NameList IS TABLE OF emp.ename%TYPE;

TYPE SalList IS TABLE OF emp.sal%TYPE;

CURSOR c1 IS SELECT ename, sal FROM emp WHERE sal> 1000;

names NameList;

sals SalList;

TYPE RecList IS TABLE OF c1%ROWTYPE;

recs RecList;

PROCEDURE print\_results IS

BEGIN

dbms\_output.put\_line('Results:');

IF names IS NULL OR names.COUNT = 0 THEN

RETURN;

END IF;

FOR i IN names.FIRST .. names.LAST LOOP

dbms\_output.put\_line(' Employee ' || names(i) || ': $' || sals(i));

END LOOP;

END;

BEGIN

dbms\_output.put\_line('--- Processing all results at once ---');

OPEN c1;

FETCH c1 BULK COLLECT INTO names, sals;

CLOSE c1;

print\_results;

dbms\_output.put\_line('--- Processing 7 rows at a time ---');

OPEN c1;

LOOP

FETCH c1 BULK COLLECT INTO names, sals LIMIT 7;

EXIT WHEN c1%NOTFOUND;

print\_results;

END LOOP;

print\_results;

CLOSE c1;

dbms\_output.put\_line('--- Fetching records rather than columns ---');

OPEN c1;

FETCH c1 BULK COLLECT INTO recs;

FOR i IN recs.FIRST .. recs.LAST

LOOP

dbms\_output.put\_line(' Employee ' || recs(i).ename || ': $' || recs(i).sal);

END LOOP;

END;

/

DECLARE

TYPE DeptRecTab IS TABLE OF dept%ROWTYPE;

dept\_recs DeptRecTab;

CURSOR c1 IS SELECT deptno, dname, loc FROM dept WHERE deptno > 10;

BEGIN

OPEN c1;

FETCH c1 BULK COLLECT INTO dept\_recs;

END;

/

### Returning

Permet d'éviter de faire une requête en plus de l'INSERT/UPDATE ou DELETE.

Ex :

declare

type tab is table of emp%rowtype index by binary\_integer;

monTab tab;

nb pls\_integer;

begin

update emp set sal = sal + 1

where comm is null

returning empno,ename,job,mgr,hiredate,sal,comm,deptno

bulk collect into monTab;

nb := monTab.count;

dbms\_output.put\_line(nb);

end ;

/

Autre exemple :

CREATE TABLE emp2 AS SELECT \* FROM emp;

DECLARE

TYPE NumList IS TABLE OF emp.empno%TYPE;

enums NumList;

TYPE NameList IS TABLE OF emp.ename%TYPE;

names NameList;

BEGIN

DELETE FROM emp2 WHERE deptno = 30

RETURNING empno, ename BULK COLLECT INTO enums, names;

dbms\_output.put\_line('Deleted ' || SQL%ROWCOUNT || ' rows:');

FOR i IN enums.FIRST .. enums.LAST LOOP

dbms\_output.put\_line('Employee #' || enums(i) || ': ' || names(i));

END LOOP;

END;

/

## Tips & Tricks PL/SQL

* Les tables PL : Ces tableaux peuvent, dans certaines conditions, est « castés » en table Oracle afin de les intégrer dans une requête.

Ex :

declare

tab mdsys.num\_array;

type monTypeDeTableau is table of integer index by binary\_integer;

monTableau monTypeDeTableau;

begin

select empno bulk collect into tab from emp;

for rec in (select t.deptno, count(\*) cnt from emp v

join dept t on t.deptno = v.deptno

group by t.deptno)

loop

monTableau(rec.deptno) := rec.cnt;

end loop;

for i in monTableau.first..monTableau.last loop

if monTableau.exists(i) then

dbms\_output.put\_line(i||': '||monTableau(i));

end if;

end loop;

end;

/

* Eviter l'appel de fonction dans une requête
  + au niveau du select : pour éviter d'appeler plusieurs fois la même fonction avec des paramètres identiques
  + au niveau de clause Where : pour la même raison et pour permettre l'accès aux indexs.
* Eviter les conversions implicites

DECLARE

n NUMBER;

c CHAR(5);

BEGIN

n := n + 15; -- converted implicitly; slow. (15 is a PLS\_INTEGER)

n := n + 15.0; -- not converted; fast (15.0 is a NUMBER)

c := 25; -- converted implicitly; slow

c := TO\_CHAR(25); -- converted explicitly; still slow

c := '25'; -- not converted; fast

END;

/

* Interdiction d'annuler toutes les exceptions (WHEN OTHERS null;)

Les exceptions les plus communes sont déjà nommées (ex NO\_DATA\_FOUND).

Pour les autres, il est possible de les nommer (PRAGMA EXCEPTION INIT)

*Lors de l'utilisation de "WHEN OTHERS", ne pas oublier de renvoyer l'erreur au programme appelant (raise;).*

declare

mon\_exception exception ;

pragma exception\_init(mon\_exception, -6502) ;

begin

select 1 into ma\_var from tab1 ;

exception

when no\_data\_found then

-- traitement

when mon\_exception then

-- autre traitement

end ;

* Bind variable

Afin de profiter au maximum du share pool pour ne pas parser inutilement des requêtes, faire attention à utiliser des variables bindées.

Par défaut, PL/SQL bind ses variables lors de l'utilisation de paramêtre

Ex :

create or replace procedure dsal(p\_empno in number) as

begin

update emp set sal=sal\*2 where empno = p\_empno;

end;

/

sauf dans le cas d'exécution dynamique

|  |  |  |
| --- | --- | --- |
| create or replace procedure dsal(p\_empno in number)  as  begin  execute immediate  'update emp set sal = sal\*2 where empno = '||p\_empno;  end; | devient | create or replace procedure dsal(p\_empno in number)  as  begin  execute immediate  'update emp set  sal = sal\*2 where empno = :x' using p\_empno;  end; |

Set serveroutput on

alter system flush shared\_pool;

alter system flush buffer\_cache;

declare

type rc is ref cursor;

l\_rc rc;

l\_dummy all\_objects.object\_name%type;

l\_start number default dbms\_utility.get\_time;

begin

for i in 1 .. 1000

loop

open l\_rc for

'select object\_name from all\_objects where object\_id = ' || i;

fetch l\_rc into l\_dummy;

close l\_rc;

dbms\_output.put\_line(l\_dummy);

end loop;

dbms\_output.put\_line (round((dbms\_utility.get\_time-l\_start)/100, 2) || ' Sec...' );

end;

/

62 Sec…

declare

type rc is ref cursor;

l\_rc rc;

l\_dummy all\_objects.object\_name%type;

l\_start number default dbms\_utility.get\_time;

begin

for i in 1 .. 1000

loop

open l\_rc for

'select object\_name from all\_objects where object\_id = :x' using i;

fetch l\_rc into l\_dummy;

close l\_rc;

dbms\_output.put\_line(l\_dummy);

end loop;

dbms\_output.put\_line (round((dbms\_utility.get\_time-l\_start)/100, 2) ||' Sec...' );

end;

/

* 1. ec…

Avec du SQL dynamique

create or replace procedure dsal(p\_empno in number) as

begin

execute immediate

'update emp set sal = sal\*2 where empno = '||p\_empno;

end;

/

create or replace procedure dsal(p\_empno in number) as

begin

execute immediate

'update emp set

sal = sal\*2 where empno = :x' using p\_empno;

end;

/

ROLLUP et CUBE permettent d'éviter de faire des calculs en PL/SQL ou d'autres requêtes SQL après récupération des données

Ex :

select decode(grouping(job),'1', ' Total', job) job,

decode(grouping(ename),'1', ' Total', ename) NOM,sum(sal) from emp

GROUP BY rollup(job, ename);

Attention à SQLDevelopper qui corrige à tort l'expression "GROUP BY" lors de l'ajout des expressions "GROUPING"

Ex :

|  |  |  |
| --- | --- | --- |
| select distinct(ma\_fonction(col1))  from tab1 | devient | select ma\_fonction(res)  from (select distinct col1 res  from tab1) |

Si tabl comporte 100 enregistrements et si les 100 valeurs de col1 sont identiques, ma\_fonction sera appelée 100 fois dans le premier cas et une seule fois dans le second.

### Sélections hiérarchiques (CONNECT\_BY)

→ permettent d'éviter de faire des calculs en PL/SQL ou d'autres requêtes SQL après récupération des données

Ex :

select lpad(level,level,' ') niveau, ch.\* from emp ch

connect by prior empno = mgr

start with mgr is null

order siblings by ch.ename;

### Divers

* utilisation de "exists" au lieu de "in (select [...])
* utilisation de "left join" au lieu de "not exists"
* attention au db\_link (penser à l'utilisation de vues ou tables temporaires)

# v$sqlarea

v$sqlarea n'est pas un outil d'analyse de sql, mais une vue système ( owner : sys ) qui contient les derniers sql exécutés sur la base .