SQL Dynamique en PL/SQL

Anne-Cécile Caron

Master MIAGE - BDA

1er trimestre 2010-2011

Introduction

- En PL/SQL, on ne peut pas utiliser d'instruction DDL.
- En PL/SQL, les requêtes SQL sont intégrées dans le programme : le langage de requête fait partie du langage de programmation
- Avantage : les requêtes sont compilées durant la compilation du programme. On n'est pas obligé d'attendre l'exécution pour vérifier leur validité.
- Inconvénient : les requêtes sont statiques (même si on peut définir des paramètres)
- Pour exécuter des instructions DDL ou des instructions DML non connues à la compilation : *SQL Dynamique*

Quand l'utiliser ?

- On a besoin de créer une table, ou de définir le rôle de l'utilisateur courant.
  ⇒ on n'est plus dans les requêtes DML, on doit utiliser SQL dynamique
- On a une requête DML qui comporte des inconnues comme l'ordre de tri, la liste des colonnes, le nom de la table, ...
  ⇒ Ce n'est plus simplement des valeurs dans la clause `where` qu'il faut compléter, on doit utiliser SQL dynamique.

```
select nom,prenom from client where id_client = ?
--> ok en SQL intégré à PL/SQL

select ? from ? where id_client=?
--> besoin de SQL dynamique
```


Deux alternatives

Il y a deux méthodes pour faire du SQL dynamique en PL/SQL

1. SQL dynamique "natif", où on place directement le code dans les blocks PL/SQL. Utilisable uniquement côté serveur.
2. Utilisation du paquetage `dbms_sql`. (on ne verra pas cette solution)

SQL dynamique natif

- L'instruction `EXECUTE IMMEDIATE` compile une requête et l'exécute (SQL dynamique ou bloc anonyme)
- L'argument est une chaîne de caractères qui contient l'instruction à exécuter.
- La chaîne peut décrire n'importe quel type d'instruction SQL ou bloc PL/SQL, sauf une requête qui ramène plusieurs lignes
- syntaxe (incomplète) :

```
EXECUTE IMMEDIATE dynamic_string
 [ INTO { define_variable [, define_variable ...]
          | record_name } ]
 [ USING [ IN | OUT | IN OUT ] bind_argument
           [, [ IN | OUT | IN OUT ] bind_argument] ... ]
```


Exemple 1

```
CREATE OR REPLACE PROCEDURE delete_rows (
   table_name IN VARCHAR2,
   condition  IN VARCHAR2 DEFAULT NULL) AS

   where_clause  VARCHAR2(100) := ' WHERE ' || condition;
   v_table       VARCHAR2(30);
BEGIN
-- first make sure that the table actually exists; if not, raise an exception
  SELECT OBJECT_NAME INTO v_table FROM USER_OBJECTS
    WHERE OBJECT_NAME = UPPER(table_name) AND OBJECT_TYPE = 'TABLE';
   IF condition IS NULL THEN where_clause := NULL; END IF;
   EXECUTE IMMEDIATE 'DELETE FROM ' || v_table || where_clause;
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('Invalid table: ' || table_name);
END;
/
BEGIN
  delete_rows('CLIENT', 'id_client = 111');
END;
/
```


clause INTO

- Permet de récupérer les résultats d'une requête (comme le `select ... into` de PL/SQL)
- Comme pour le `select ... into`, cette clause n'est valable que pour les requêtes qui ramènent 1 ligne.
- *On peut utiliser la clause* `bulk collect into` *pour récupérer plusieurs lignes dans une collection*

```
create function row_count (tab_name VARCHAR2) RETURN NUMBER AS
   rows NUMBER;
begin
   EXECUTE IMMEDIATE 'select count(*) from ' || tab_name into rows;
   return rows;
end;

-- row_count('employee') donne le nombre de lignes de la table employee
```


clause USING

- Permet de passer des paramètres effectifs.
- A l'exécution, les paramètres de la clause `USING` remplacent les "trous" formés par les paramètres formels dans la chaîne de caractères constituant la requête SQL dynamique.
- Les modes `IN`, `OUT`, `IN OUT` ont la même signification que pour les paramètres d'une procédure PL/SQL.
- Le mode par défaut est `IN`
- La requête est une chaîne de caractères → bien souvent, on peut se passer de paramètres et utiliser la concaténation MAIS (comme les requêtes préparées en JDBC) utiliser une seule requête paramétrée est plus efficace que reconstruire une nouvelle requête par concaténation.

Exemple 2

```
CREATE OR REPLACE PROCEDURE raise_emp_salary (column_value NUMBER,
                            emp_column VARCHAR2, amount NUMBER) IS
   v_column VARCHAR2(30);
   sql_stmt  VARCHAR2(200);
BEGIN
-- determine if a valid column name has been given as input
  SELECT COLUMN_NAME INTO v_column FROM USER_TAB_COLS
    WHERE TABLE_NAME = 'EMPLOYEES' AND COLUMN_NAME = emp_column;
  sql_stmt := 'UPDATE employees SET salary = salary + :1 WHERE '
              || v_column || ' = :2';
  EXECUTE IMMEDIATE sql_stmt USING amount, column_value;
  IF SQL%ROWCOUNT > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Salaries have been updated for: ' || emp_column
                         || ' = ' || column_value);
  END IF;
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('Invalid Column: ' || emp_column);
END raise_emp_salary;
/
```


Exemple 3

```
CREATE PROCEDURE ma_procedure (
   param1 IN OUT NUMBER,
   param2  IN VARCHAR2 ) AS
BEGIN
   ...
END;
/
DECLARE
   plsql_block VARCHAR2(500);
   p1  NUMBER(4) := 3 ;
   p2   VARCHAR2(30) := 'toto';
BEGIN
   plsql_block := 'BEGIN ma_procedure(:a, :b); END;';
   EXECUTE IMMEDIATE plsql_block
      USING in out p1, in p2;
   dbms_output.put_line(p1);
END;
/
```


Les curseurs

- Comme la requête n'est pas connue, on utilise une variable de type `REF CURSOR`.
- Un `REF CURSOR` peut être vu comme un pointeur de curseur. Il permet de passer des curseurs en paramètres de procédures, ou de manière générale, de manipuler des curseurs qui ne sont pas liés à 1 requête précise.
- on utilise les instructions suivantes pour manipuler la variable curseur, :
  1. `OPEN var_cur FOR requete_dyn`,
  2. `FETCH var_cur INTO un_record`,
  3. `CLOSE var_cur`
  4. mais PAS la syntaxe abrégée
     `for un_rec in le_curseur loop ... end loop`

Exemple

```
DECLARE
   TYPE EmpCurTyp IS REF CURSOR;
   emp_cv   EmpCurTyp;
   emp_rec  employees%ROWTYPE;
   sql_stmt VARCHAR2(200);
   v_job   VARCHAR2(10) := 'ST_CLERK';
BEGIN
   sql_stmt := 'SELECT * FROM employees WHERE job_id = :j';
   OPEN emp_cv FOR sql_stmt USING v_job;
   LOOP
     FETCH emp_cv INTO emp_rec;
     EXIT WHEN emp_cv%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('Name: ' || emp_rec.last_name || ' Job Id: ' ||
                          emp_rec.job_id);
   END LOOP;
   CLOSE emp_cv;
END;
/
```

bulk dynamic SQL

tout ce qu'on a vu sur "bulk SQL" reste valable en SQL dynamique

```
DECLARE
   TYPE EmpCurTyp IS REF CURSOR;
   TYPE NumList IS TABLE OF NUMBER;
   TYPE NameList IS TABLE OF VARCHAR2(25);
   emp_cv EmpCurTyp;
   empids NumList;
   enames NameList;
   sals   NumList;
BEGIN
   OPEN emp_cv FOR 'SELECT employee_id, last_name FROM employees';
   FETCH emp_cv BULK COLLECT INTO empids, enames;
   CLOSE emp_cv;

   EXECUTE IMMEDIATE 'SELECT salary FROM employees'
      BULK COLLECT INTO sals;
END;
```