Les Curseurs

Définition

Pour pouvoir exécuter une requête, le système de gestion de base de données alloue dynamiquement et gère une zone mémoire appelée **contexte**.
Le contexte contient les informations nécessaires à l'exécution de la requête SQL. A savoir, le résultat de l'interprétation de la requête, l'adresse des variables hôtes, le statut d'exécution de la requête, etc.
Cette zone peut être manipulée par l'intermédiaire d'un identificateur appelé un **curseur**. Il s'agit en fait d'un pointeur sur le contexte.

Un contexte référencé par un curseur est donc :

- une structure de données en mémoire,
- capable de stocker le résultat d'une requête.

Cette structure est particulièrement bien adaptée (et même indispensable) à la manipulation de données en mémoire lorsque le **nombre de lignes ramenées** par un SELECT est **inconnu**, ce qui est généralement le cas dans la pratique.

PL/SQL utilise deux types de curseurs :

- les curseurs **nommés** pour traiter les requêtes multi lignes
- les curseurs **non nommés** – accessible avec le mot clé SQL – utilisés automatiquement pour les autres commandes SQL

Traitement des curseurs nommés

A chaque traitement d'une commande SQL, PL/SQL ouvre une zone de contexte pour exécuter la commande et stocker les données. Le curseur nommé permet :

- de nommer cette zone de contexte,
- d'accéder aux données,
- de contrôler le traitement si nécessaire.

Pour chaque requête SELECT qui peut retourner plus d'un enregistrement, on associe un curseur nommé et on s'y réfère dans la suite du programme pour traiter les lignes les unes après les autres.
On peut maintenir autant de curseurs ouverts simultanément que les ressources système le permettent.

Pour utiliser un curseur nommé, il faut :

- le **déclarer** comme une variable à l'aide du mot clé **CURSOR**
- l'**ouvrir**
- le **parcourir**
- le **refermer** une fois le traitement terminé afin de libérer les ressources utilisées.

Déclaration du curseur

La déclaration d'un curseur consiste à le nommer et à lui associer une requête. La syntaxe est la suivante :

```
CURSOR c_nomcurseur [(nom _param1 type [, nom_param2 type]…)]  IS
   requêteSELECT ;
```

Les paramètres du curseur ne peuvent être utilisés que pour la requête SELECT du curseur.

- Curseur simple : exemple de déclaration

```
DECLARE
   CURSOR c_employe IS
      SELECT emp_nom, emp_dep_no FROM exe_employe WHERE emp_salaire > 2000;
```

- Curseur paramétré : exemple de déclaration

```
DECLARE
   CURSOR c_employe_du_dept (v_dep_no exe_dept.dep_no%TYPE) IS
      SELECT * FROM exe_employe WHERE emp_dep_no = v_dep_no ;
```

Parcours d'un curseur avec une boucle FOR *(foreach, parcours exhaustif)*

Il est possible de simplifier l'utilisation d'un curseur dans la partie traitement en utilisant une boucle FOR. En effet, ce type de boucle déclare implicitement une variable de type record permettant de stocker chaque enregistrement retourné par la requête du curseur. De plus, cette boucle ouvre, parcours, puis ferme le curseur automatiquement. Pas besoin de faire « manuellement » l'OPEN, FETCH et CLOSE.

-- Curseur non paramétré avec boucle FOR

```
DECLARE
  CURSOR c_employe IS
     SELECT emp_nom, emp_dep_no
        FROM exe_employe
        WHERE emp_salaire > 2000;

BEGIN
   -- Parcourir les employés et afficher leur nom
  FOR r_employe IN c_employe LOOP
      dbms_output.put_line(r_employe.emp_nom);
  END LOOP;
END;
```

-- Curseur paramétré avec boucle FOR

```
DECLARE
  CURSOR c_employe (v_dep_no
       exe_employe.emp_dep_no%TYPE) IS
    SELECT emp_nom, emp_dep_no
       FROM exe_employe
       WHERE emp_dep_no = v_dept_no;
BEGIN
   -- Parcourir les employés et afficher leur nom
  FOR r_employe IN c_employe(3) LOOP
      dbms_output.put_line(r_employe.emp_nom);
  END LOOP;
END;
```

Utilisation d'une boucle WHILE pour parcourir le curseur

Lorsque vous ne souhaitez pas faire un parcours exhaustif de tout le curseur, mais que vous voulez rajouter une condition d'arrêt, ou que vous voulez effectuer un traitement spécifique du premier / du dernier enregistrement, utilisez une boucle WHILE afin de faire un parcours « manuel » du curseur.

Ouverture du curseur

L'ouverture du curseur se fait :

- soit explicitement avec la commande OPEN,
- soit implicitement si on utilise la commande FOR pour parcourir le curseur (voir plus haut).

Le curseur se positionne automatiquement sur la première ligne retournée par la requête SQL du curseur.

```
BEGIN
   OPEN c_employe    ou    OPEN c_employe_du_dept(parametre) ;
```

Parcours du curseur

Lors de son ouverture, le curseur pointe sur la première ligne. La commande FETCH permet :

- de stocker les valeurs contenues dans le curseur dans des variables déclarées,
- de trouver la ligne suivante.

```
FETCH c_employe INTO v_emp_nom, v_emp_dep_no;
```

Fermeture du curseur

Pour libérer les ressources du curseur une fois le traitement fini, on utilise la commande CLOSE.

```
CLOSE c_employe ;
```

Exemple de parcours d'un curseur avec une boucle WHILE

Exemple complet de PL/SQL utilisant un curseur paramétré

-- Bloc anonyme affichant le nom des employés du département n° 3 *(numéro passé en paramètre au curseur)* :

```
DECLARE
   CURSOR c_employe_du_dept (v_dep_no exe_dept.dep_no%TYPE) IS
      SELECT * FROM exe_employe WHERE emp_dep_no = v_dep_no ;
   r_employe  exe_employe%ROWTYPE;
BEGIN
   OPEN c_employe_du_dept (3);
   FETCH c_employe_du_dept INTO r_employe;
   WHILE c_employe_du_dept%found LOOP
      dbms_output.put_line(r_employe.emp_prenom || ' ' || r_employe.emp_nom);
      FETCH c_employe_du_dept INTO r_employe;
   END LOOP;
   CLOSE c_employe_du_dept;
END;
```

Attributs du curseur

PL/SQL fournit des attributs pour les curseurs qui sont :

%ISOPEN : retourne vrai si le curseur est ouvert

```
IF c_employe%ISOPEN THEN
   CLOSE c_employe;
ELSE
   OPEN c_employe;
END IF;
```

%ROWCOUNT : nombre de lignes déjà parcourues avec la commande FETCH

%NOTFOUND, %FOUND : état du dernier FETCH

%NOTFOUND retourne Vrai si le dernier FETCH a échoué à cause de la non disponibilité de données (fin du curseur).

Curseurs non-nommés : SQL%

La base de données ouvre une zone de contexte pour le traitement de chaque ordre SQL. Ceci permet, y compris pour une commande qui n'est pas un SELECT, d'accéder aux informations de la zone contexte par la syntaxe SQL%ATTRIBUT.

Exemple d'utilisation d'un curseur non-nommé :

```
BEGIN
   UPDATE exe_employe
      SET emp_salaire = emp_salaire + 100
    WHERE emp_salaire < 2000;

   IF SQL%NOTFOUND THEN            -- Traitement du cas particulier
      dbms_output.put_line('Personne n''a été augmenté !');
   ELSE
      dbms_output.put_line('augmentation effectuée...');
   END IF;
END;
```

Les Curseurs Avancés

Les Curseurs Dynamiques (SYS_REFCURSOR)

Les curseurs dynamiques ont la particularité de pouvoir être compilés sans que la requête associée ne soit connue. C'est lors de l'exécution que la requête sera attribuée au curseur. Ainsi, en utilisant ce type de curseur il est possible de construire la requête SQL par programmation.

Création d'un type de curseur

Pour créer un curseur dynamique, il faut utiliser SYS_REFCURSOR, ou créer un type REF CURSOR pour déclarer une variable de ce type. La requête sera attribuée dynamiquement au curseur et exécutée lors de l'ouverture de celui-ci.

```
DECLARE
     c_curseur  SYS_REFCURSOR ;
-- ou :
     TYPE ty_curseur IS REF CURSOR [RETURN varchar];
     c_curseur  ty_curseur ;
```

Ouverture du curseur

Lors de l'ouverture du curseur, une requête lui est affectée. La chaîne contenant la requête SQL peut être construite.

```
DECLARE
   c_curseur  SYS_REFCURSOR ;
   v_sql      VARCHAR2(200) ;
BEGIN
   v_sql := 'SELECT colonnes FROM nom_table' ;
   …
   OPEN c_curseur FOR v_sql ;
   …
END ;
```

Curseurs dynamiques fortement ou faiblement typé

Lors de la déclaration du type pour le curseur dynamique, le type de retour est optionnel. S'il est présent, le compilateur va vérifier les types si cela est possible. Ce curseur dynamique dit à typage fort, présente l'avantage d'être contrôlé lors de la compilation.
L'absence de type de retour, s'il comporte l'inconvénient de ne pas être contrôlé, permet une programmation très souple puisque la structure du curseur est totalement dynamique. Ce second type de curseur dynamique dit à typage faible nécessite une programmation très propre car si lors du FETCH les variables hôtes ne sont pas correctement typées par rapport au curseur, l'exception ROWTYPE_MISMATCH sera déclenchée.

Exemples de Curseur Dynamique

```
PROCEDURE AffiListeEmployes(tri_par_dep IN BOOLEAN) IS
   -- Création d'un curseur dynamique pour remplacer les 2 requêtes suivantes :
   --    CURSOR c_employes_par_nom IS SELECT * FROM exe_employe ORDER BY emp_nom;
   --    CURSOR c_employes_par_dep IS SELECT * FROM exe_employe ORDER BY emp_dep_no, emp_nom;
   c_employe  SYS_REFCURSOR;
   r_employe  exe_employe%ROWTYPE;
   v_sql      VARCHAR2(200);
BEGIN
   -- création de la requête selon les paramètres reçus (ordre de tri)
   v_sql := 'SELECT * FROM exe_employe ORDER BY ';
   IF tri_par_dep THEN v_sql := v_sql || 'emp_dep_no, '; END IF;
   v_sql := v_sql || 'emp_nom';

   OPEN c_employe FOR v_sql;       -- ouverture du curseur ==> exécution de la requête

   FETCH c_employe INTO r_employe;  -- parcours du curseur
   WHILE c_employe%FOUND LOOP
      dbms_output.put_line(r_employe.emp_prenom || ' ' || r_employe.emp_nom || '...');
      FETCH c_employe INTO r_employe;
   END LOOP;

   CLOSE c_employe;
END AffiListeEmployes;
/
BEGIN   -- Test de la procédure :
   AffiListeEmployes(FALSE);    dbms_output.put_line('--------------------------');
   AffiListeEmployes(TRUE);
END;
```

Curseur modifiable

Il est possible de mettre à jour les données d'une table directement à travers un curseur qui lui est associé.
Il faut pour cela déclarer le curseur avec une clause FOR UPDATE, afin qu'Oracle puisse gérer l'accès multi-utilisateur aux données de la table (exclusif row locks).

Exemple de déclaration d'un curseur pour la mise à jour

```
DECLARE
   CURSOR c_employe IS
      SELECT emp_nom, emp_prenom FROM exe_employe WHERE emp_dep_no > 20
         FOR UPDATE OF emp_salaire;
```

Les verrous seront posés lors de l'exécution de la commande OPEN et relâchés sur le COMMIT marquant la fin de la transaction. Lors de la manipulation des données dans le curseur, il faut utiliser la clause WHERE CURRENT OF pour référencer le curseur actif.

Exemple de mise à jour de données à partir d'un curseur

```
BEGIN
   OPEN c_employe;
   FETCH c_employe INTO v_emp_nom, v_emp_prenom;
   WHILE c_employe%FOUND LOOP
      dbms_output.put(v_emp_prenom || ' ' || v_emp_nom);
      IF LOWER(v_emp_prenom) = 'alain' THEN
         UPDATE exe_employe
             SET emp_salaire = emp_salaire * 1.2
           WHERE CURRENT OF c_employe;    -- update du record ramené par le dernier fetch
         dbms_output.put_line('   ==> augmentation de salaire');
      ELSE
         dbms_output.put_line('   ==> pas de chance');
      END IF;
      FETCH c_employe INTO v_emp_nom, v_emp_prenom;
   END LOOP;
   CLOSE c_employe;
END;
```

Lorsque les données sont manipulées sur une requête multi-table, il est possible de ne verrouiller que certaines tables en utilisant la clause FOR UPDATE OF.