

PL/SQL - 3

Objectifs du cours

- Curseurs
- Séquences

Curseurs

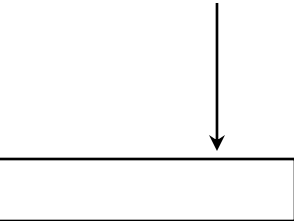
- Qu'est-ce qu'un curseur
- Quand utiliser des curseurs avec PL/SQL
- Comment utiliser un curseur
 - curseurs implicites
 - curseurs explicites
 - déclaration
 - ouverture
 - accès
 - fermeture

Qu'est-ce qu'un curseur

- Jusqu'à présent, l'utilisation de l'instruction SELECT est limitée aux requêtes renvoyant une et une seule valeur
Toute requête renvoyant un résultat de cardinalité différente de 1 aboutit à un echec
- Définition : un curseur est un mécanisme permettant d'utiliser une instruction SELECT renvoyant un nombre quelconque de tuples
- Note : à toute requête SQL exécutée par le serveur Oracle est associée un curseur

Qu'est-ce qu'un curseur

- Un curseur peut être considéré comme une fenêtre sur l'ensemble des résultats d'une requête

NumVille	Date	Température	Curseur
A512	25-MAR-1997	26	
A512	26-MAR-1997	28	
A512	27-MAR-1997	29	
A512	28-MAR-1997	40	
A512	30-MAR-1997	27	
A512	01-APR-1997	25	

- On distingue deux types de curseurs :
 - curseurs implicites : déclarés implicitement par PL/SQL lors de tout SELECT
 - curseurs explicites : créés par le programmeur

Quand utiliser un curseur

- Les curseurs doivent être utilisés lorsque l'on désire utiliser la totalité du résultat d'une requête SELECT
- En particulier si cette requête renvoie un résultat de cardinalité > 1
- Le curseur pointe toujours sur une valeur du résultat de la requête à laquelle il est associé
- Un curseur permet :
 - de garder trace d'une requête SELECT
 - de parcourir tuple par tuple le résultat d'un SELECT
 - d'accéder à la totalité du résultat d'un SELECT

Quand utiliser un curseur

```
DECLARE
    noDept_v Departement.noDept%TYPE;
    lieu_v Departement.lieu%TYPE;
BEGIN
    SELECT      noDept, lieu
              INTO  noDept_v, lieu_v
    FROM Departement
    WHERE      nomDept = 'VENTES';

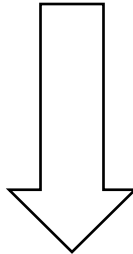
    ...
END ;
```

Mais s'il y a
plusieurs départements
des ventes ?



Comment utiliser un curseur

- Protocole d'utilisation :
 - Déclaration
 - Ouverture
 - Utilisation
 - Fermeture
- Attributs associés aux curseurs
- Fonctions associées aux curseurs
- Parcours de curseurs



Comment utiliser un curseur - Exemple

DECLARE

CURSOR departementVentes_v IS

SELECT *

FROM Departement

WHERE nomDept = 'VENTES' ;

unDepartement_v Departement%ROWTYPE ;

compteur_v number := 0 ;

BEGIN

OPEN departementVentes_v ;

LOOP

FETCH departementVentes_v INTO unDepartement_v ;

EXIT WHEN departementVentes_v%NOTFOUND ;

compteur_v := compteur_v + 1 ;

END LOOP ;

CLOSE departementVentes_v ;

END ;

← déclaration

← ouverture

← utilisation

← utilisation

← fermeture

Déclaration d'un curseur

- Syntaxe :
CURSOR <nom de curseur> IS
 <instruction SELECT> ;
- Notes :
 - ne pas utiliser de clause INTO dans l'instruction SELECT
 - si vous voulez manipuler les tuples dans un ordre spécifique, utilisez la clause ORDER BY dans la requête (voir cours SQL)
 - on peut déclarer autant de curseurs que l'on veut
 - le contenu du curseur n'est pas calculé au moment de sa déclaration, mais au moment de son ouverture

Attributs associés aux curseurs

- Exemple :
EXIT WHEN departementVentes_v%NOTFOUND ;
- Utilisez les attributs de curseurs pour tester le résultat de vos requêtes SQL
 - %ROWCOUNT
nombre de tuples pointés par le curseur depuis son ouverture
 - %FOUND
booléen, vaut TRUE si le curseur pointe vers un tuple
 - %NOTFOUND
booléen, vaut TRUE si le curseur pointe derrière le dernier tuple
 - %ISOPEN
booléen indiquant si le curseur est ouvert ou fermé (par défaut, les curseurs implicites sont toujours fermés à la fin de la requête)
- Note :
utilisez 'SQL' comme nom de curseur pour identifier l'état d'un curseur implicite (i.e. créé automatiquement lors d'une requête classique)

Fonctions associées aux curseurs

- Exemple :
 FETCH departementVentes_v INTO unDepartement_v ;
- Utilisez les fonctions associée aux curseurs pour accéder au contenu du résultat
 - OPEN <nom de curseur>
 ouverture du curseur, exécution la requête associée
 positionnement du pointeur juste avant le premier tuple du résultat
 si le résultat de la requête est vide, aucune erreur n'est levée
 - FETCH ...
 déplacement du pointeur vers le prochain tuple du curseur
 affectation du nouveau tuple pointé à une variable
 - CLOSE <nom de curseur>
 fermeture du curseur
 désallocation de la mémoire associée
 perte des données associées au curseur
 (=> fermer systématiquement les curseurs après utilisation)

Fonctions associées aux curseurs - FETCH

- Syntaxe :
FETCH <nom de curseur> INTO [[<variable1>, <variable2>, ...]
| <nom de record>] ;
- Notes :
 - seulement pour les curseurs explicites
 - inclure le même nombre de variables dans la clause INTO qu'il y a d'attributs dans le SELECT associé au curseur
 - mettre les variables dans le bon ordre
 - alternativement, utiliser un record ayant le type adapté
 - utiliser le test %FOUND ou %NOTFOUND pour voir si la fonction FETCH a permis d'atteindre un nouveau tuple, ou si l'on est à la fin du curseur
 - avant le premier appel à la fonction FETCH, l'attribut %NOTFOUND du curseur vaut NULL => pensez-y pour éviter des boucles infinies

Parcours d'un curseur

- Syntaxe :

FOR <nom de record> IN <nom de curseur> LOOP
 <instructions>

...

END LOOP ;

- Notes :

- ne pas déclarer le record, celui-ci l'est implicitement
- ne pas ouvrir ni fermer le curseur, les fonctions OPEN, FETCH et CLOSE sont déclenchées automatiquement
- il est possible de mettre directement la requête à la place du nom de curseur; dans ce cas, il n'est plus utile de déclarer de curseur

Parcours d'un curseur - Exemple

```
DECLARE
    CURSOR departementVentes_v IS
        SELECT      *
        FROM        Departement
        WHERE        nomDept = 'VENTES' ;
    compteur_v number := 0 ;
BEGIN
    FOR chaqueDepartement_v IN departementVentes_v LOOP
        compteur_v := compteur_v + 1 ;
    END LOOP ;
END ;
```

Parcours d'un curseur - Exemple

```
DECLARE
    compteur_v number := 0 ;
BEGIN
    FOR chaqueDepartement_v IN (      SELECT      *
                                      FROM      Departement
                                      WHERE      nomDept = 'VENTES'
                                ) LOOP
        compteur_v := compteur_v + 1 ;
    END LOOP ;
END ;
```


Curseurs paramétrés

- Syntaxe :
 - Déclaration
CURSOR <nom de curseur>
 ([<nom de paramètre> [IN] <type> [{:= | DEFAULT} <valeur>]) *
IS
 <instruction SELECT utilisant les paramètres> ;
 - Ouverture
OPEN <nom de curseur> (<valeur1>, <valeur2>, ...)
 - Boucle
FOR <nom de record> IN <nom de curseur> (<valeur1>, <valeur2>, ...) LOOP
 <instructions>

 ...
END LOOP ;
 - Fermeture
CLOSE <nom de curseur>

Curseurs paramétrés - Exemple

```
DECLARE
    CURSOR departement_v (nomDept_p varchar2(15)) IS
        SELECT      *
        FROM        Departement
        WHERE        nomDept = nomDept_p ;
    compteur_v number := 0 ;
BEGIN
    FOR chaqueDepartement_v IN departement_v ('VENTES') LOOP
        compteur_v := compteur_v + 1 ;
    END LOOP ;
END ;
```

Modification d'un curseur

- Exemple :

DECLARE

CURSOR departementVentes_v (nomDept_p varchar2) IS

SELECT *

FROM Departement

WHERE nomDept = nomDept_p

FOR UPDATE ;

BEGIN

FOR chaqueDepartement_v IN departementVentes_v ('VENTES') LOOP

UPDATE Departement SET noDept = 123

WHERE CURRENT OF departementVentes_v ;

END LOOP ;

END ;

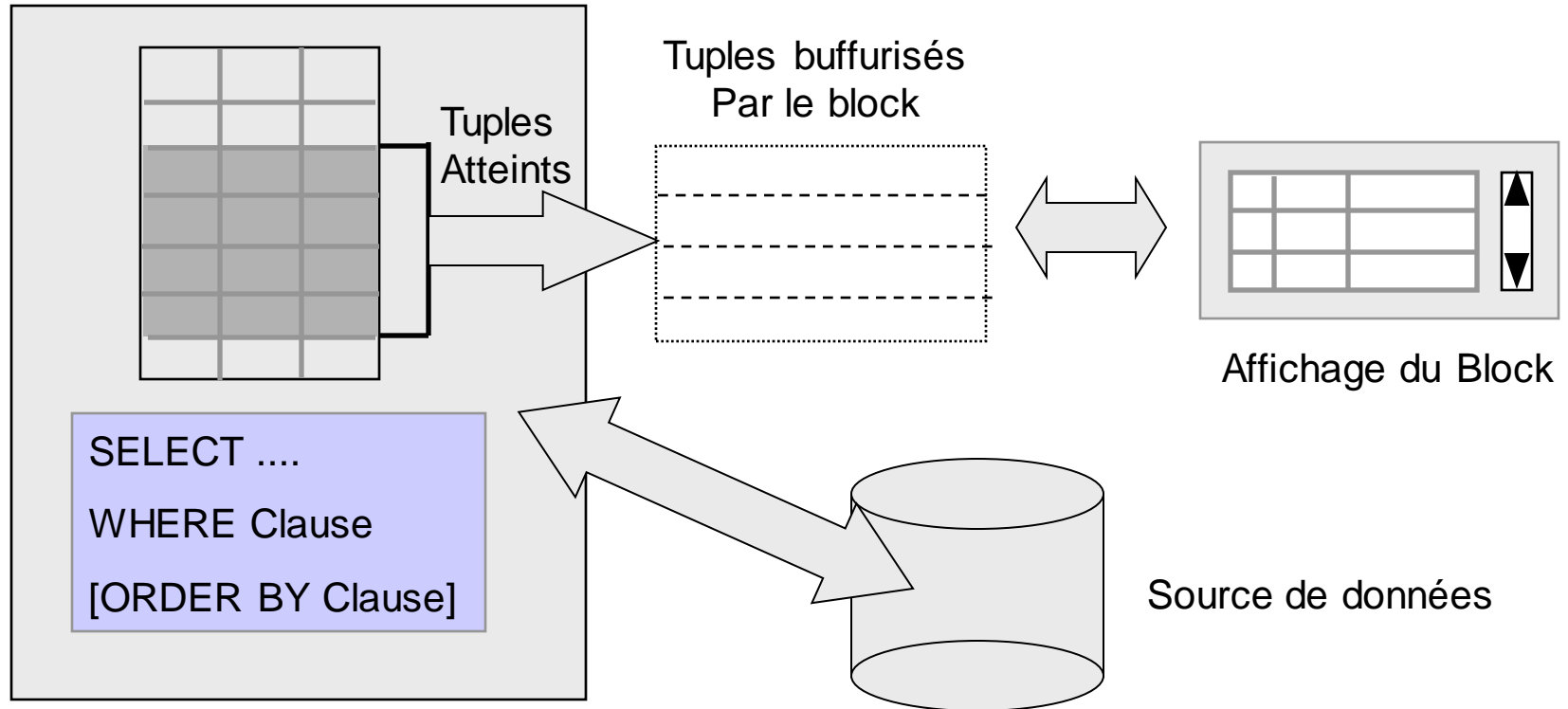
- Notes :

- ne pas oublier le 'FOR UPDATE' pour déclarer la modification
- utiliser la clause 'UPDATE ... WHERE' pour effectuer la modification

Et dans Developer/2000 ?

La notion de curseur apparaît à travers les blocks de données

CURSEUR



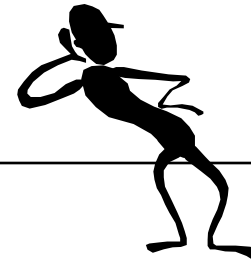
Et dans Developer/2000 ?

- On peut spécifier le curseur du block à travers les propriétés DATABASE du block et de ses items
- On peut se déplacer par programmation dans un curseur affiché avec un block de données (i.e. en fait, un block de données ayant la propriété DATABASE positionnée à YES) en :
 - Se positionnant sur le block : **GO_BLOCK('<nom du block>')**
 - Utilisant la procédure **NEXT_RECORD** Pour avancer
 - Utilisant la procédure **PREVIOUS_RECORD** Pour reculer
- Attention aux propriétés de navigation des blocks et de leurs éléments !!!

Séquences

- Similaires à des variables entières qui s'auto-incrémentent
- Particulièrement utiles pour la génération des valeurs de clefs
- Création d'une séquence dans SQLPlus :
create sequence empNo_seq start with 1 increment 1
- Récupération d'une valeur par requête :
select empNo_seq.nextval into toto_v from dual;
- Utilisation de la séquence au moment de l'insertion d'un tuple dans une table :
insert into Emp_t values (empNo_seq.nexval, 'Joe Black');
- Utilisation de la séquence au moment de la création d'un record dans un block de données :
propriété INITIAL VALUE = :SEQUENCE.empNo_seq.nextval

Exercices



- Créez une table :
Meteo (NumVille, Date, Température)
- Ecrivez deux procédures utilisant des curseurs :
 - l'une qui transforme, dans la table, les températures de degrés Fahrenheit en degrés Celcius
 - l'une qui transforme, dans la table, les températures de degrés Celcius en degrés Fahrenheit

$$\mathbf{D^{\circ}\text{Celcius} = 5/9 (D^{\circ}\text{Fahrenheit} - 32)}$$

- Peuplez la table d'une dizaine de valeurs et testez vos deux procédures
 - vérifiez par vous même les résultats obtenus
 - appliquez les deux procédures à la même table et vérifiez que le contenu de la table n'a pas changé

Exercices

- Créez une table Note(NumEtud, NumMat, note) et créez un formulaire permettant d'accéder à son contenu tuple par tuple. Vous ajouterez deux boutons pour l'exploration du contenu de la table : l'un permettant d'avancer, l'autre de reculer
- Ajoutez au formulaire ci dessus un bouton et un champ non BD permettant d'afficher les notes en utilisant le format A/B/C/D au lieu de 1, 2, 3 .. 20
- Créez à nouveau le formulaire ci-dessus, mais cette fois en spécifiant un block de données et des éléments ayant la propriété DATABASE positionnée à NO