

# Développement avec PL/SQL (2)

SYSINF-SES-CUI  
UNIVERSITE DE GENEVE

Thang LE DINH  
Email: [thang@cui.unige.ch](mailto:thang@cui.unige.ch)  
Web: <http://cui.unige.ch/~thang>

## Contenu

1. Introduction
  2. Variables et constantes
  3. Types de données
  4. Conditions et répétitions
  5. **Intégration des commandes SQL**
  6. **Les curseurs**
  7. Gestion des erreurs
  8. Procédures et fonctions
- Partie 1
- Partie 2
- Partie 3

## 5. Intégration des commandes SQL

- 5.1 Le *langage de manipulation de données* (LMD)
- 5.2 La gestion des transactions

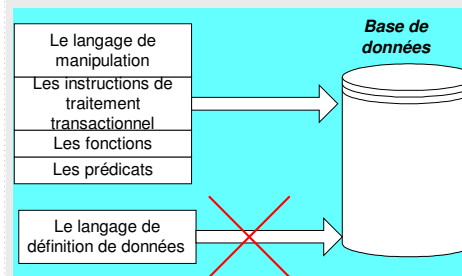
05/04/2004

Développement avec PL/SQL 2

3

### 5.1 Le LMD

- Les instructions SQL sont à la base de tout le langage PL/SQL
  - Par exemple: le langage **LMD** SQL, les instructions de traitement transactionnel, les fonctions et les prédicats.
- On ne peut pas utiliser les instructions **LDD** (*langage de définition de données*) avec un bloc PL/SQL
  - Parce qu'elles peuvent dépasser le champ du traitement transactionnel.



05/04/2004

Développement avec PL/SQL 2

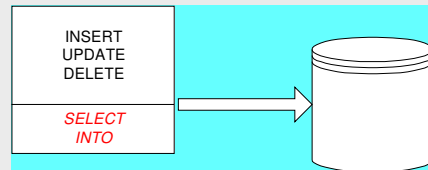
4

## 5.1 Le LMD

### ■ Le langage LMD SQL:

Pour manipuler les données via les commandes LMD:

- **INSERT** : insérer un tuple;
- **UPDATE**: mettre à jour des valeurs d'un tuple;
- **DELETE**: supprimer un tuple.
- **SELECT INTO**: sélectionner les données



05/04/2004

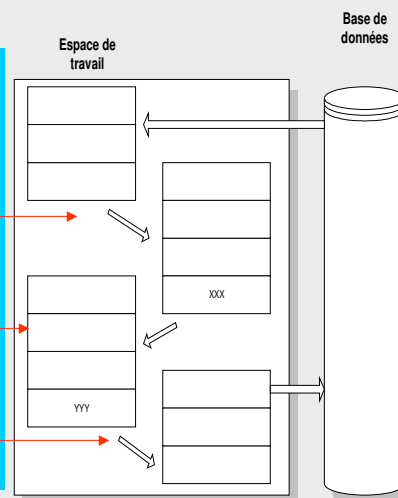
Développement avec PL/SQL 2

5

## 5.1 Le LMD

### Exemple 5.1:

```
BEGIN
/* Manipulation des tuples de la table
chauffeur CH (NCH, CHAUFFEUR) */
-- Insérer un nouveau tuple
INSERT INTO ch
VALUES (5, 'Mr XXX');
-- Modifier ce tuple
UPDATE ch SET chauffeur = 'Mr YYY'
WHERE nch = 5;
-- Supprimer ce tuple
DELETE FROM ch WHERE nch = 5 ;
END;
```



05/04/2004

Développement avec PL/SQL 2

6

## 5.1 Le LMD

### ■ Le langage LMD SQL:

Pour *sélectionner les données* via la commande SELECT

#### ■ Syntaxe:

```
SELECT les_colonnes INTO les_variables  
FROM les_tables  
[WHERE les_conditions]  
[GROUP BY les_colonnes]  
[HAVING les_conditions]
```

- Une instruction **SELECT .. INTO** doit renvoyer exactement une ligne
- Il est possible de lancer des instructions SELECT multi-lignes à l'aide de **curseurs**

05/04/2004

Développement avec PL/SQL 2

7

## 5.1 Le LMD

- *Exemple 5.2: Faites l'entrée une période (par exemple: de 01.01.2001 à 30.06.2001) et affichez le total moyen des nombres de kilomètres de tous les trajets pendant cette période*

```
DECLARE  
total_moyen    NUMBER;  
de_jour DATE := '&De';  
a_jour DATE := '&A';  
  
BEGIN  
  
    SELECT AVG(nbkkm) INTO total_moyen FROM trajet WHERE  
    datetrajet BETWEEN de_jour AND a_jour ;  
  
    DBMS_OUTPUT.PUT_LINE('Total moyen: ' ||  
    TO_CHAR(total_moyen));  
END ;  
/
```

05/04/2004

Développement avec PL/SQL 2

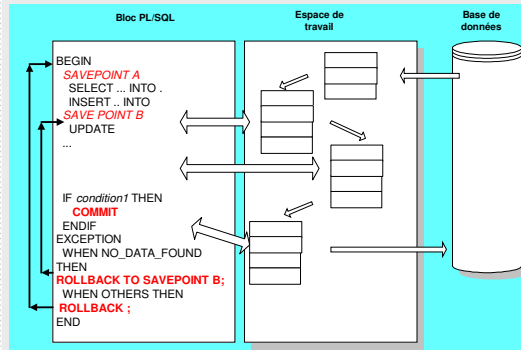
8

## 5.2 La gestion des transactions

### ■ Une transaction:

- Est un **groupe d'instructions** de manipulation de données SQL;
- Traitée par Oracle comme **une seule unité**;
- Si bien que tous les effets des instructions **s'appliquent** ou **s'annulent** simultanément.

- Les instructions de traitement transactionnel: **COMMIT**, **ROLLBACK**, **SAVEPOINT**



05/04/2004

Développement avec PL/SQL 2

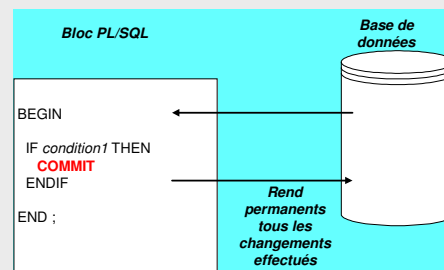
9

## 5.2 La gestion des transactions

### ■ COMMIT

→ La persistance

- Syntaxe: **COMMIT**
- COMMIT rend permanents tous les changements effectués durant la transaction courante;
- Tant qu'on a validé les changements, les autres utilisateurs peuvent les voir



05/04/2004

Développement avec PL/SQL 2

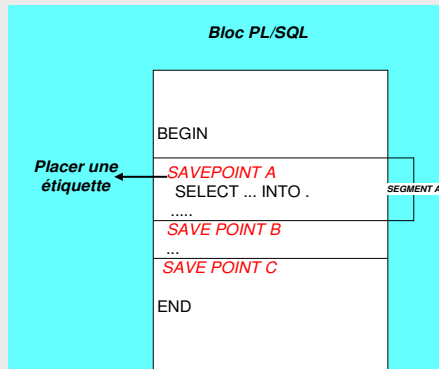
10

## 5.2 La gestion des transactions

### ■ SAVEPOINT

→ Les segments

- Syntaxe: **SAVEPOINT** **<nom\_point\_sauvegarde>**
- SAVEPOINT place une étiquette dans le traitement d'une transaction
- SAVEPOINT est utilisées conjointement avec l'instruction ROLLBACK



05/04/2004

Développement avec PL/SQL 2

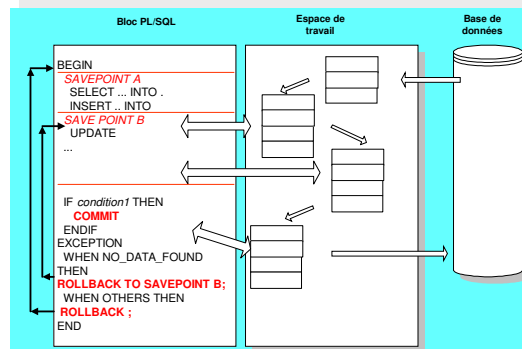
11

## 5.2 La gestion des transactions

### ■ ROLLBACK

→ Undo

- Syntaxe:  
**ROLLBACK [TO [SAVEPOINT] <nom\_point\_sauvegarde> ]**
- ROLLBACK annule toutes les actions appliquées depuis le début de cette transaction ou depuis le **<nom\_point\_sauvegarde>**



05/04/2004

Développement avec PL/SQL 2

12

## 5.2 La gestion des transactions

### ■ Exemple 5.3: Modifiez le prix d'une réparation donnée

```

DECLARE
test CHAR(1) := 'N';
m_norep reparation.norep%TYPE;
m_px reparation.px%TYPE;
m_px1 reparation.px%TYPE;
m_px2 reparation.px%TYPE;
BEGIN
-- Entrez les variables
m_norep := &Numero_reparation ;
m_px := &prix_reparation ;
test := '&This_is_a_test_Y_N' ;
-- Mettre à jour
UPDATE reparation
SET px = m_px
WHERE norep = m_norep ;

```

```

-- Affichez les valeurs à l'espace de travail
SELECT px INTO m_px1 FROM reparation
WHERE norep = m_norep ;
DBMS_OUTPUT.PUT_LINE('Prix 1: '||m_px1);
IF (test = 'Y') OR (test = 'y') THEN
    ROLLBACK ;
ELSE
    COMMIT ;
END IF ;
-- Affichez les valeurs à la table REPARATION
SELECT px INTO m_px2 FROM reparation
WHERE norep = m_norep ;
DBMS_OUTPUT.PUT_LINE('Prix 2: '||m_px2);
END ;

```

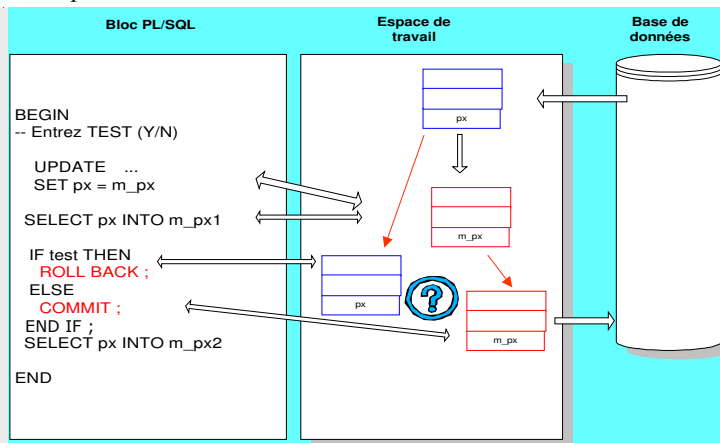
05/04/2004

Développement avec PL/SQL 2

13

## 5.2 La gestion des transactions

### ■ Exemple 5.3:



05/04/2004

Développement avec PL/SQL 2

14

## 5.2 La gestion des transactions

### ■ Exemple 5.3:

Oracle SQL\*Plus

```
SQL> select * from reparation ;
```

| NOREP | NOU | NOG | TYPERE | PX   | KMCPT |
|-------|-----|-----|--------|------|-------|
| 551   | 11  | 901 | A      | 5200 | 4000  |
| 552   | 11  | 902 | B      | 4000 | 4800  |
| 553   | 11  | 901 | B      | 3000 | 5200  |
| 554   | 12  | 902 | C      | 3000 | 3000  |
| 555   | 12  | 903 | A      | 6000 | 3600  |

```
SQL> @exemple5_3 ;
Enter value for numero_reparation: 554
old 9: m_norep := &Numero_reparation ;
new 9: m_norep := 554 ;
Enter value for prix_reparation: 12000
old 10: m_px := &prix_reparation ;
new 10: m_px := 12000 ;
Enter value for this_is_a_test_y_n: Y
old 11: test := '&This_is_a_test_V_N' ;
new 11: test := 'Y' ;
Prix 1: 12000
Prix 2:
PL/SQL procedure successfully completed.
SQL> |
```

05/04/2004 Développement avec PL/SQL 2

**ROLLBACK** →

15

## 5.2 La gestion des transactions

Oracle SQL\*Plus

```
SQL> select * from reparation ;
```

| NOREP | NOU | NOG | TYPERE | PX   | KMCPT |
|-------|-----|-----|--------|------|-------|
| 551   | 11  | 901 | A      | 5200 | 4000  |
| 552   | 11  | 902 | B      | 4000 | 4800  |
| 553   | 11  | 901 | B      | 3000 | 5200  |
| 554   | 12  | 902 | C      | 3000 | 3000  |
| 555   | 12  | 903 | A      | 6000 | 3600  |

```
SQL> @exemple5_3 ;
Enter value for numero_reparation: 554
old 9: m_norep := &Numero_reparation ;
new 9: m_norep := 554 ;
Enter value for prix_reparation: 12000
old 10: m_px := &prix_reparation ;
new 10: m_px := 12000 ;
Enter value for this_is_a_test_y_n: N
old 11: test := '&This_is_a_test_V_N' ;
new 11: test := 'N' ;
Prix 1: 12000
Prix 2: 12000
PL/SQL procedure successfully completed.
SQL> select * from reparation where norep = 554 ;
```

| NOREP | NOU | NOG | TYPERE | PX    | KMCPT |
|-------|-----|-----|--------|-------|-------|
| 554   | 12  | 902 | C      | 12000 | 3000  |

05/04/2004 Développement avec PL/SQL 2

**COMMIT** →

16



## Contenu

1. Introduction
  2. Variables et constantes
  3. Types de données
  4. Conditions et répétitions
  5. Intégration des commandes SQL
  6. Les curseurs
  7. Gestion des erreurs
  8. Procédures et fonctions
- Partie 1
- Partie 2
- Partie 3

05/04/2004

Développement avec PL/SQL 2

17

## 6. Les curseurs

- Introduction
- Les curseurs explicites
- Les curseurs implicites

05/04/2004

Développement avec PL/SQL 2

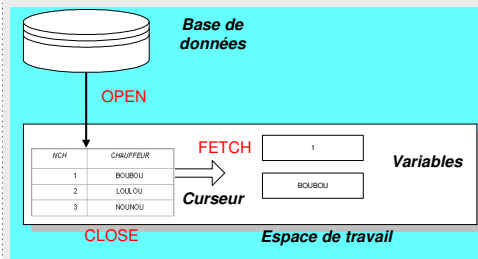
18

## 6.1 Introduction

- Pour chaque instruction SQL, Oracle ouvre un *domaine privé* pour la traiter:

→ **curseur**

- Un curseur permet de *nommer le domaine privé* et d'accéder aux informations stockées.



05/04/2004

Développement avec PL/SQL 2

19

## 6.1 Introduction

- Il y a deux types de curseur: *implicite* et *explicite*
- Les **curseurs implicites** sont établis automatiquement INSERT, UPDATE, DELETE et SELECT..INTO (à une seul ligne)
- Les **curseurs explicites** doivent être déclarés SELECT..INTO (aux multi-lignes)

05/04/2004

Développement avec PL/SQL 2

20

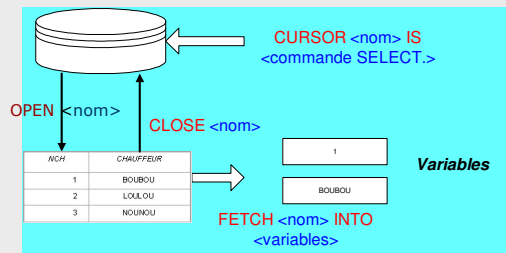
## 6.2 Les curseurs explicites

- Pour déclarer et utiliser un **curseur explicite**:

- Déclaration du curseur
- Ouverture du curseur
- Recherche des données provenant du curseur
- Fermeture du curseur

- Pour traiter:

- Attributs de curseur
- Les boucles FOR du curseur



05/04/2004

Développement avec PL/SQL 2

21

## 6.2 Les curseurs explicites

- **Déclaration du curseur**

- Syntaxe:

```
CURSOR <nom_curseur>
IS <instruction SELECT>
```

DECLARE

```
m_date DATE := '&Entrez_une_date';
```

```
CURSOR cur_trajet IS
SELECT notraj, villedep, villearr
FROM trajet
WHERE datetrajet = m_date ;
```

BEGIN

.....

END ;

/

05/04/2004

Développement avec PL/SQL 2

22

## 6.2 Les curseurs explicites

### ■ Ouverture du curseur

#### ■ Syntaxe:

**OPEN** <nom\_curseur>

#### DECLARE

m\_date DATE := '&Entrez\_une\_date';

CURSOR cur\_trajet IS

SELECT notraj, villedep, villearr

FROM trajet

WHERE datetraj = m\_date ;

#### BEGIN

OPEN cur\_trajet;

#### END ;

/

05/04/2004

Développement avec PL/SQL 2

23

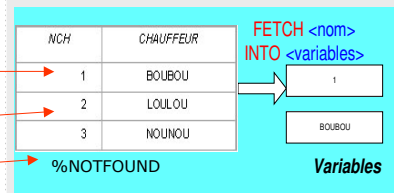
## 6.2 Les curseurs explicites

### ■ Recherche des données provenant du curseur

- PL/SQL demande la première ligne renvoyée et la traite...
- Demande la ligne suivante et la traite ...
- Et ainsi de suite jusqu'à la dernière ligne

### ■ L'instruction FETCH

- FETCH place le contenu de la ligne courante dans les variables locales.
- Syntaxe:  
**FETCH** <nom\_curseur> INTO  
    <les\_variables>



05/04/2004

Développement avec PL/SQL 2

24

## 6.2 Les curseurs explicites

### DECLARE

```
m_date DATE := '&Entrez_une_date';
m_notraj trajet.notraj%TYPE ;
m_villedep trajet.villedep%TYPE ;
m_villearr trajet.villearr%TYPE ;
CURSOR cur_trajet IS
  SELECT notraj, villedep, villearr FROM trajet
  WHERE datetraj = m_date ;
```

### BEGIN

```
OPEN cur_trajet;
LOOP
  FETCH cur_trajet INTO m_notraj, m_villedep, m_villearr;
  -- Traiter les données
END LOOP;
CLOSE cur_trajet
```

### END ;

05/04/2004

Développement avec PL/SQL 2

25

## 6.2 Les curseurs explicites

### ■ Fermeture du curseur

#### ■ Syntaxe:

**CLOSE** <nom\_curseur>

### DECLARE

```
m_date DATE := 'Entrez_une_date';
CURSOR cur_trajet IS
  SELECT notraj, villedep, villearr
  FROM trajet
  WHERE datetraj = m_date ;
```

### BEGIN

```
OPEN cur_trajet;
-- Traiter le curseur
CLOSE cur_trajet ;
```

### END ;

05/04/2004

Développement avec PL/SQL 2

26

## 6.2 Les curseurs explicites

- **Exemple 6.1:** Affichez la liste de tous les trajets effectués pendant une journée

```

DECLARE
m_date DATE := '&Entrez_une_date' ;
m_notraj trajet.notraj%TYPE ;
m_villedep trajet.villedep%TYPE ;
m_villearr trajet.villearr%TYPE ;
CURSOR cur_trajet IS
    SELECT notraj, villedep, villearr
    FROM trajet
    WHERE datetraj = m_date ;
BEGIN
    OPEN cur_trajet;
    LOOP
        FETCH cur_trajet
        INTO m_notraj, m_villedep, m_villearr;
        EXIT WHEN
            cur_trajet%NOTFOUND ;
        -- traiter les données
        DBMS_OUTPUT.PUT_LINE
            ('No trajet: '||m_notraj
             ||' départ: '||m_villedep
             ||' arrivée: '||m_villearr);
    END LOOP;
    CLOSE cur_trajet;
END ;
/
    
```

05/04/2004

Développement avec PL/SQL 2

27

## 6.2 Les curseurs explicites

- **Exemple 6.1:** Affichez la liste de tous les trajets pendant une journée

```

Oracle SQL*Plus
File Edit Search Options Help

SQL> select * from trajet ;

   NOTRAJ  VILLEDEP      VILLEARR      DATETRAJE      NBKM
-----
    101 GENEVE          ZURICH        01-JAN-00        180
    102 GENEVE          LAUSANNE      15-FEB-00         60
    103 GENEVE          BERN          10-APR-00        120
    108 LAUSANNE        BERN          20-AUG-00         60
    110 LAUSANNE        ZURICH        02-JAN-01        120
    201 GENEVE          ZURICH        01-JAN-00        180
    202 GENEVE          LAUSANNE      16-FEB-00         60
    203 GENEVE          BERN          19-APR-00        120
    208 LAUSANNE        BERN          29-AUG-00         60
    209 BERN            GENEVE        29-JAN-01        120

10 rows selected.

SQL> @exemple5_4;
Enter value for entrez_une_date: 01-JAN-00
old 2: m_date DATE := '&Entrez_une_date' ;
new 2: m_date DATE := '01-JAN-00' ;
No trajet:101 départ: GENEVE      arrivée: ZURICH
No trajet:201 départ: GENEVE      arrivée: ZURICH

PL/SQL procedure successfully completed.

SQL>
    
```

05/04/2004

Développement avec PL/SQL 2

28

## 6.2 Les curseurs explicites

### ■ Attributs de curseur

- %NOTFOUND
- %FOUND
- %ROWCOUNT
- %ISOPEN

### ■ Pour utiliser:

- Exemple:
  - Les curseurs explicites  
→ cur\_trajet%FOUND
  - Les curseurs implicites  
→ SQL%FOUND

### ■ Exemple 6.2:

```
DECLARE
m_nch      ch.nch%TYPE :=
            '&nom_chauffeur';
m_chauffeur ch.chauffeur%TYPE ;

BEGIN
SELECT chauffeur INTO m_chauffeur
FROM ch WHERE m_nch = nch ;

IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Il y a ce
chauffeur!') ;
END IF ;
END ;
```

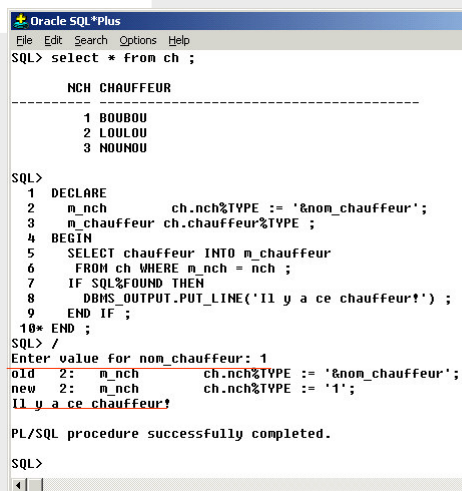
05/04/2004

Développement avec PL/SQL 2

29

## 6.2 Les curseurs explicites

### ■ Exemple 6.2:



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> select * from ch ;

      NCH CHAUFFEUR
-----
1 BOUBOU
2 LOULOU
3 NOUOU

SQL>
1 DECLARE
2   m_nch      ch.nch%TYPE := '&nom_chauffeur';
3   m_chauffeur ch.chauffeur%TYPE ;
4 BEGIN
5   SELECT chauffeur INTO m_chauffeur
6   FROM ch WHERE m_nch = nch ;
7   IF SQL%FOUND THEN
8     DBMS_OUTPUT.PUT_LINE('Il y a ce chauffeur!') ;
9   END IF ;
10* END ;
SQL> /
Enter value for nom_chauffeur: 1
old 2:  m_nch      ch.nch%TYPE := '&nom_chauffeur';
new 2:  m_nch      ch.nch%TYPE := '1';
Il y a ce chauffeur!

PL/SQL procedure successfully completed.

SQL>
```

05/04/2004

Développement avec PL/SQL 2

30

## 6.2 Les curseurs explicites

### ■ %NOTFOUND

- Utilisez l'attribut **%NOTFOUND** pour déterminer quand il ne reste plus de lignes à rechercher

```
BEGIN
OPEN cur_trajet;
LOOP
  FETCH cur_trajet INTO
    m_notraj, m_villedep, m_villearr;

  EXIT WHEN cur_trajet%NOTFOUND;
  -- Traiter le curseur
END LOOP;

CLOSE cur_trajet ;
END ;

/
```

05/04/2004

Développement avec PL/SQL 2

31

## 6.2 Les curseurs explicites

### ■ %FOUND

- Est le contraire logique de %NOTFOUND

```
LOOP
  FETCH cur_trajet INTO
    m_notraj, m_villedep, m_villearr;

  IF cur_trajet%FOUND THEN
    -- Traiter le curseur
  ELSE
    EXIT
  END IF ;

  ....
END LOOP
```

05/04/2004

Développement avec PL/SQL 2

32



## 6.2 Les curseurs explicites

■ **%ROWCOUNT**: Utilisez pour agir lorsqu'un nombre spécifié de lignes a été recherché

```
LOOP
  FETCH cur_trajet INTO
    m_notraj, m_villedep, m_villearr;

  EXIT WHEN (cur_trajet%NOTFOUND)
    OR (cur_trajet%ROWCOUNT > 10);
  .....
END LOOP;
```

05/04/2004

Développement avec PL/SQL 2

33

## 6.2 Les curseurs explicites

■ **%ISOPEN**:  
retournez l'état du curseur

```
BEGIN
  ....
  IF NOT (cur_trajet%ISOPEN) THEN
    OPEN cur_trajet ;
  END IF ;
  LOOP
    FETCH cur_trajet INTO
      m_notraj, m_villedep, m_villearr;
    EXIT WHEN (cur_trajet%NOTFOUND);
    .....
  END LOOP
END ;
```

05/04/2004

Développement avec PL/SQL 2

34

## 6.2 Les curseurs explicites

- Les **boucles FOR** du curseur:  
s'exécutent une fois pour chaque  
ligne contenue dans le curseur

- Syntaxe:

```
FOR <nom_enregistrement>  
IN <nom_curseur> LOOP
```

```
-- Traiter une ligne
```

```
END LOOP ;
```

Un nom\_curseur  
OPEN implicite est  
exécuté

Une commande  
FETCH implicite est  
exécuté

S'il ne reste plus de  
lignes, un  
nom\_curseur CLOSE  
implicite est exécuté

05/04/2004

Développement avec PL/SQL 2

35

## 6.2 Les curseurs explicites

### ■ Exemple 6.3:

```
DECLARE  
m_date DATE :=  
'&Entrez_une_date' ;  
m_nombre_trajets INTEGER;  
CURSOR cur_trajet IS  
SELECT notraj, villedep, villearr  
FROM trajet  
WHERE datetraj = m_date ;
```

```
BEGIN  
m_nombre_trajets := 0 ;  
FOR rec_trajet IN cur_trajet LOOP  
-- traiter les données  
DBMS_OUTPUT.PUT_LINE  
( 'No trajet: ' || rec_trajet.notraj  
|| ' départ: ' || rec_trajet.villedep  
|| ' arrivée: ' || rec_trajet.villearr );  
m_nombre_trajets :=  
m_nombre_trajets + 1;  
END LOOP;  
DBMS_OUTPUT.PUT_LINE  
( 'Total: ' || m_nombre_trajets || ' trajets' );  
END ;
```

05/04/2004

Développement avec PL/SQL 2

36

## 6.2 Les curseurs explicites

### ■ Exemple 6.3:

```

1 DECLARE
2   n_date DATE := '&Entrez_une_date' ;
3   n_nombre_trajets INTEGER;
4   CURSOR cur_trajet IS
5     SELECT notraj, villedep, villearr, datetraj
6     FROM trajet
7     WHERE datetraj = n_date ;
8 BEGIN
9   n_nombre_trajets := 0 ;
10  FOR rec_trajet IN cur_trajet LOOP
11    -- traiter les données
12    DBMS_OUTPUT.PUT_LINE
13      ('No trajet: '||rec_trajet.notraj
14       ||' départ: '||rec_trajet.villedep
15       ||' arrivée: '||rec_trajet.villearr);
16    n_nombre_trajets := n_nombre_trajets + 1;
17  END LOOP;
18  DBMS_OUTPUT.PUT_LINE
19    ('Total: '||n_nombre_trajets||' trajets');
20* END ;
21 /
Entrez une valeur pour entrez_une_date : 01/01/00
ancien 2 : n_date DATE := '&Entrez_une_date' ;
nouveau 2 : n_date DATE := '01/01/00' ;
No trajet: 101 départ: GENEVE          arrivée: ZURICH
No trajet: 201 départ: GENEVE          arrivée: ZURICH
Total: 2 trajets
Procédure PL/SQL terminée avec succès.

```

05/04/2004

Développement avec PL/SQL 2

37

## 6.2 Les curseurs explicites

### ■ Paramètres de curseur

- Les curseurs acceptent aussi des paramètres

#### Syntaxe:

- **CURSOR** <nom\_curseur>  
(nom\_para type\_para)  
**IS** <instruction SELECT>
- **OPEN** <nom\_curseur>  
(paramètre)
- **FOR** <nom\_enregistrement>  
**IN** <nom\_curseur> (paramètre)  
**LOOP** ....

#### DECLARE

```

CURSOR cur_trajet IS
  SELECT notraj, villedep, villearr, datetraj
  FROM trajet
  WHERE villedep = 'GENEVE' ;

```

Généralisez des curseurs avec des paramètres

#### DECLARE

```

CURSOR cur_trajet (p_ville VARCHAR) IS
  SELECT notraj, villearr, datetraj
  FROM trajet
  WHERE villedep = UPPER(p_ville) ;

```

#### BEGIN

```

OPEN cur_trajet ('GENEVE');
.....
END ;

```

05/04/2004

Développement avec PL/SQL 2

38

## 6.2 Les curseurs explicites

- Exemple 6.4: Affichez la liste de tous les trajets partant d' une ville donnée

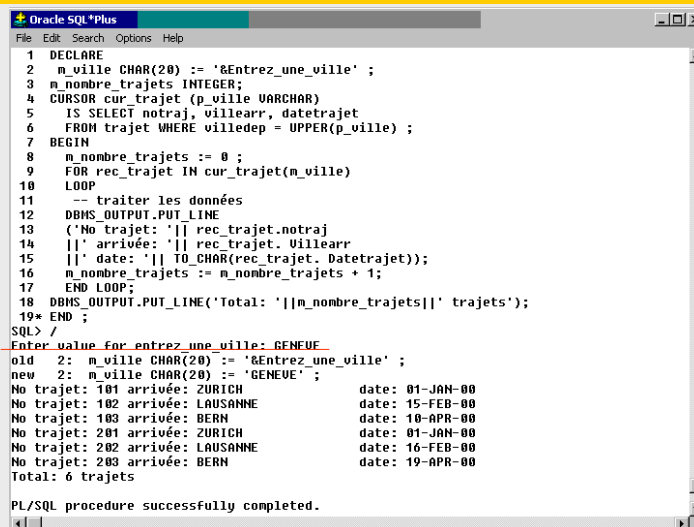
```
DECLARE
    m_ville VARCHAR(20) :=
    '&Entrez_une_ville';
    m_nombre_trajets INTEGER := 0;
    CURSOR cur_trajet (p_ville VARCHAR)
    IS
        SELECT notraj, villearr, datetraj
        FROM trajet
        WHERE TRIM(villedep) =
        UPPER(p_ville) ;
BEGIN
    FOR rec_trajet IN cur_trajet
        (TRIM(m_ville)) LOOP
        -- traiter les données
        DBMS_OUTPUT.PUT_LINE('No
trajet: ' || rec_trajet.notraj || ' arrivée: ' ||
rec_trajet.Villearr || ' date: ' ||
TO_CHAR(rec_trajet.Datetraj));
        m_nombre_trajets :=
m_nombre_trajets + 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE
('Total: ' || m_nombre_trajets || ' trajets');
END ;/
```

05/04/2004

Développement avec PL/SQL 2

39

## 6.2 Les curseurs explicites



```
Oracle SQL*Plus
File Edit Search Options Help
1 DECLARE
2   m_ville CHAR(20) := '&Entrez_une_ville' ;
3   m_nombre_trajets INTEGER;
4   CURSOR cur_trajet (p_ville VARCHAR)
5   IS SELECT notraj, villearr, datetraj
6   FROM trajet WHERE villedep = UPPER(p_ville) ;
7 BEGIN
8   m_nombre_trajets := 0 ;
9   FOR rec_trajet IN cur_trajet(m_ville)
10  LOOP
11     -- traiter les données
12     DBMS_OUTPUT.PUT_LINE
13     ('No trajet: ' || rec_trajet.notraj
14     || ' arrivée: ' || rec_trajet.Villearr
15     || ' date: ' || TO_CHAR(rec_trajet.Datetraj));
16     m_nombre_trajets := m_nombre_trajets + 1;
17  END LOOP;
18  DBMS_OUTPUT.PUT_LINE('Total: ' || m_nombre_trajets || ' trajets');
19* END ;
SQL> /
Enter value for entrez_une_ville: GENEVE
old 2: m_ville CHAR(20) := '&Entrez_une_ville' ;
new 2: m_ville CHAR(20) := 'GENEVE';
No trajet: 101 arrivée: ZURICH          date: 01-JAN-00
No trajet: 102 arrivée: LAUSANNE       date: 15-FEB-00
No trajet: 103 arrivée: BERN           date: 10-APR-00
No trajet: 201 arrivée: ZURICH          date: 01-JAN-00
No trajet: 202 arrivée: LAUSANNE       date: 16-FEB-00
No trajet: 203 arrivée: BERN           date: 19-APR-00
Total: 6 trajets
PL/SQL procedure successfully completed.
```

05/04/2004

Développement avec PL/SQL 2

40