

# Développement avec PL/SQL (3)

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

## 7. Gestion des erreurs

### ■ Les erreurs?

```

1 DECLARE
2   TYPE ch_type IS RECORD
3     (ch_code NUMBER(3) ,
4      ch_nom CHAR(40));
5   un_chauffeur ch_type ;
6 BEGIN
7   -- Entrez un nouveau chauffeur
8   un_chauffeur.ch_code := &Numero_chauffeur ;
9   SELECT chauffeur INTO un_chauffeur.ch_nom
10  FROM ch WHERE nch = un_chauffeur.ch_code ;
11  DBMS_OUTPUT.PUT_LINE('Non de ce chauffeur: ' || un_chauffeur.ch_nom);
12* END ;
SQL> /
Enter value for numero_chauffeur: 0
old 8:  un_chauffeur.ch_code := &Numero_chauffeur ;
new 8:  un_chauffeur.ch_code := 0 ;
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 9

```

Erreur	Nom de l'exception
ORA-01422	TOO_MANY_ROWS
<b>ORA-01403</b>	<b>NO_DATA_FOUND</b>
ORA-01001	INVALID_CURSOR
ORA-06502	VALUE_ERROR
ORA-01476	ZERO_DIVIDE
ORA-00001	DUP_VAL_ON_INDEX
ORA-06511	CURSOR_ALREADY_OPEN

**ORA-01403: no data found**

**Cause:** In a host language program, all records have been fetched. The return code from the fetch was +4, indicating that all records have been returned from the SQL query.

**Action:** Terminate processing for the SELECT statement.

13/05/2005

Développement avec PL/SQL

3

## 7. Gestion des erreurs

### ■ Gestion des erreurs?

Erreur	Nom de l'exception
ORA-01422	TOO_MANY_ROWS
<b>ORA-01403</b>	<b>NO_DATA_FOUND</b>
ORA-01001	INVALID_CURSOR
ORA-06502	VALUE_ERROR
ORA-01476	ZERO_DIVIDE
ORA-00001	DUP_VAL_ON_INDEX
ORA-06511	CURSOR_ALREADY_OPEN

```

BEGIN
...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;

  WHEN TOO_MANY_ROWS THEN
    statement1;

  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;

```

13/05/2005

Développement avec PL/SQL

4

## 7. Gestion des erreurs

- Introduction
- Exceptions internes prédéfinies
- Exceptions définies par l'utilisateur
- Les fonctions propres à PL/SQL

13/05/2005

Développement avec PL/SQL

5

### 7.1 Introduction

- Description des exceptions
  - Dans PL/SQL, les **erreurs** sont appelées **exception**
  - Une **section de gestion d'exceptions** est une séquence d'instructions à traiter dès qu'une exception donnée de produit
  - Lorsqu'une section de gestion d'erreurs a exécuté sa procédure, le traitement du bloc se termine
- Il existe deux types d'exceptions:
  - Les **exceptions internes prédéfinies**
  - Les **exceptions définies par l'utilisateur**

13/05/2005

Développement avec PL/SQL

6

## 7.2 Exceptions internes prédéfinies

- Les exceptions internes prédéfinies les plus courantes:

<i>Erreur</i>	<i>Nom de l'exception</i>
ORA-01422	TOO_MANY_ROWS
ORA-01403	NO_DATA_FOUND
ORA-01001	INVALID_CURSOR
ORA-06502	VALUE_ERROR
ORA-01476	ZERO_DIVIDE
ORA-00001	DUP_VAL_ON_INDEX
ORA-06511	CURSOR_ALREADY_OPEN

- Les exceptions internes sont générées automatiquement (implicitement)

La liste complète : [http://cui.unige.ch/DOC/oracle8/DOC/errmsg803/A54625\\_01/toc.htm](http://cui.unige.ch/DOC/oracle8/DOC/errmsg803/A54625_01/toc.htm)

13/05/2005

Développement avec PL/SQL

7

## 7.2 Exceptions internes prédéfinies

- Déclaration des sections de gestion d'exceptions

- La partie de gestion d'exceptions d'un bloc PL/SQL doit commencer par le mot-clé **EXCEPTION** et figurer à la fin du bloc

- Syntaxe:

```
WHEN <nom de l'exception>  
[OR <nom de l'exception>...]  
THEN <séquence d'instructions>
```

13/05/2005

Développement avec PL/SQL

8

## 7.2 Exceptions internes prédéfinies

Exemple 7.1: Affichez un chauffeur

```
DECLARE
  TYPE ch_type IS RECORD
    (ch_code NUMBER(3) , ch_nom CHAR(40)) ;
  un_chauffeur ch_type ;
BEGIN
  un_chauffeur.ch_code := &Numero_chauffeur ;

  SELECT chauffeur INTO un_chauffeur.ch_nom
    FROM ch WHERE nch = un_chauffeur.ch_code ;

  DBMS_OUTPUT.PUT_LINE('Nom de ce chauffeur:'||
    un_chauffeur.ch_nom);

EXCEPTION
WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('Il n'y a pas ce chauffeur');
END ;
```

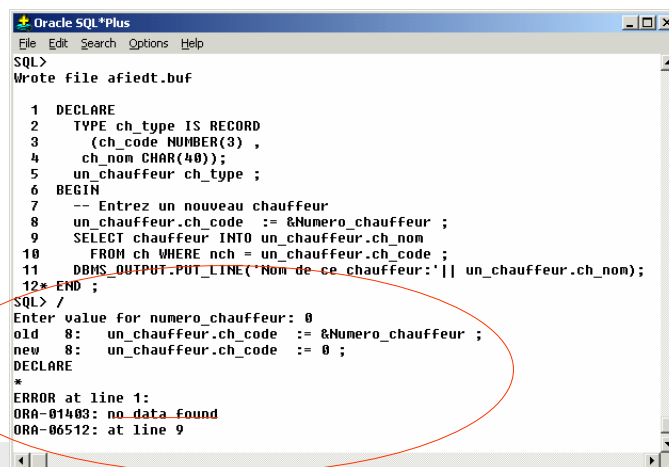
13/05/2005

Développement avec PL/SQL

9

## 7.2 Exceptions internes prédéfinies

Exemple 7.1: Affichez un chauffeur



```
Oracle SQL*Plus
File Edit Search Options Help
SQL>
Wrote file afiedt.buf

1 DECLARE
2   TYPE ch_type IS RECORD
3     (ch_code NUMBER(3) ,
4      ch_nom CHAR(40));
5   un_chauffeur ch_type ;
6 BEGIN
7   -- Entrez un nouveau chauffeur
8   un_chauffeur.ch_code := &Numero_chauffeur ;
9   SELECT chauffeur INTO un_chauffeur.ch_nom
10    FROM ch WHERE nch = un_chauffeur.ch_code ;
11   DBMS_OUTPUT.PUT_LINE('Nom de ce chauffeur:'|| un_chauffeur.ch_nom);
12*END ;
SQL> /
Enter value for numero_chauffeur: 8
old 8:  un_chauffeur.ch_code := &Numero_chauffeur ;
new 8:  un_chauffeur.ch_code := 8 ;
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 9
```

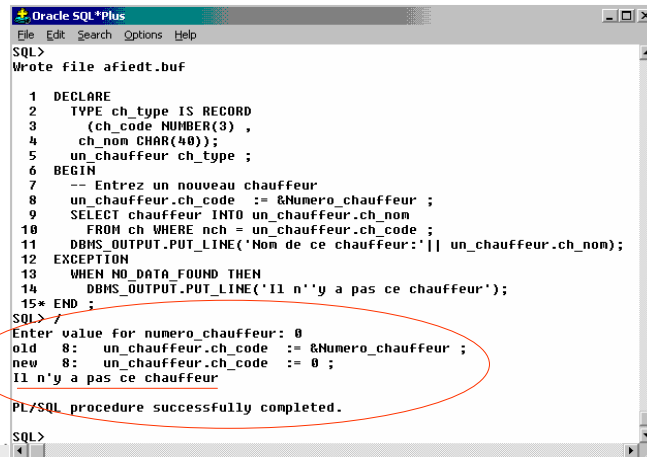
13/05/2005

Développement avec PL/SQL

10

## 7.2 Exceptions internes prédéfinies

Exemple 7.1: Affichez un chauffeur



```
Oracle SQL*Plus
File Edit Search Options Help
SQL>
Wrote file afiedt.buf

 1 DECLARE
 2   TYPE ch_type IS RECORD
 3     (ch_code NUMBER(3) ,
 4      ch_nom CHAR(40));
 5   un_chauffeur ch_type ;
 6 BEGIN
 7   -- Entrez un nouveau chauffeur
 8   un_chauffeur.ch_code := &Numero_chauffeur ;
 9   SELECT chauffeur INTO un_chauffeur.ch_nom
10     FROM ch WHERE nch = un_chauffeur.ch_code ;
11   DBMS_OUTPUT.PUT_LINE('Nom de ce chauffeur: ' || un_chauffeur.ch_nom);
12 EXCEPTION
13   WHEN NO_DATA_FOUND THEN
14     DBMS_OUTPUT.PUT_LINE('Il n''y a pas ce chauffeur');
15* END ;
SQL>
Enter value for numero_chauffeur: 8
old 8:   un_chauffeur.ch_code := &Numero_chauffeur ;
new 8:   un_chauffeur.ch_code := 8 ;
Il n'y a pas ce chauffeur

PL/SQL procedure successfully completed.
SQL>
```

13/05/2005

Développement avec PL/SQL

11

## 7.2 Exceptions internes prédéfinies

- On peut utiliser le mot-clé **OTHERS** pour traiter toutes les exceptions dont le bloc ne spécifie pas.
- Et le mot-clé **OR** pour réunir les noms d'exception.

### EXCEPTION

```
WHEN NO_DATA_FOUND OR TOO_MANY_ROWS THEN
  DBMS_OUTPUT.PUT_LINE('Il y a de problème avec ce
chauffeur');
```

```
WHEN ZERO_DIVIDE THEN
  DBMS_OUTPUT.PUT_LINE('Divide zero!');
```

```
WHEN OTHERS THEN ROLLBACK ;
  DBMS_OUTPUT.PUT_LINE('Il y a des erreurs. Annulez !');
```

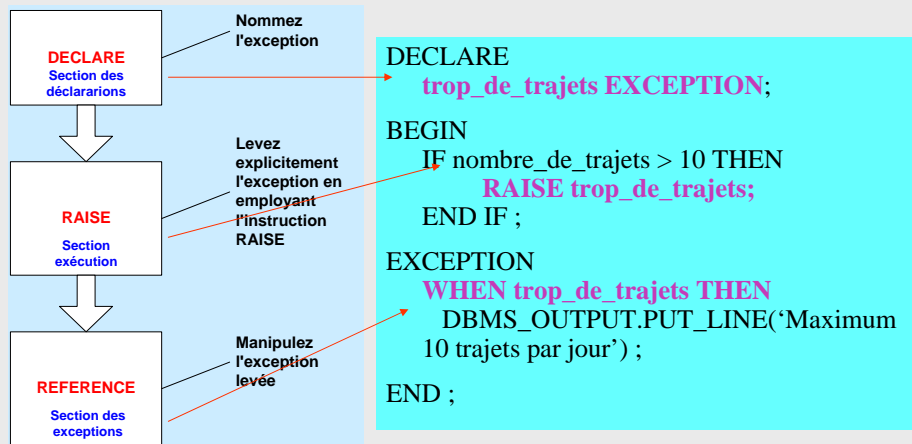
13/05/2005

Développement avec PL/SQL

12

## 7.3 Exception définie par l'utilisateur

- On peut définir nos propres exceptions  
→ *les exceptions définies par l'utilisateur*



13/05/2005

Développement avec PL/SQL

13

## 7.3 Exception définie par l'utilisateur

Exemple 7.2: Maximum 10 trajets par jour ?

```

DECLARE
  trop_de_trajets EXCEPTION;
  m_date DATE :=
    '&Entrez_une_date';
  nombre_de_trajets INTEGER;
BEGIN
  SELECT COUNT(*) INTO
    nombre_de_trajets FROM trajet
    WHERE datetrajet = m_date ;
  IF nombre_de_trajets > 10 THEN
    RAISE trop_de_trajets;
  END IF ;
  -- continuez
  --
EXCEPTION
  WHEN trop_de_trajets THEN
    DBMS_OUTPUT.PUT_LINE('Maxi
    mum 10 trajets par jour') ;
END ;
  
```

13/05/2005

Développement avec PL/SQL

14

## 7.4 Les fonctions propres à PL/SQL

- Quand on veut obtenir *plus d'informations* sur l'erreur,
  - par exemple, *son numéro* ou *son message* d'erreur Oracle
- **les fonctions de signalisation des erreurs**
- Les **fonctions propres** à PL/SQL
  - Utilisation de la signalisation des erreurs
  - Description de **EXCEPTION\_INIT**
  - Affectation d'un nom à une erreur Oracle

13/05/2005

Développement avec PL/SQL

15

## 7.4 Les fonctions propres à PL/SQL

- Deux fonctions pour la signalisation des erreurs:
  - **SQLCODE** retourne le *numéro* de l'erreur Oracle
  - **SQLERRM** retourne le *message* d'erreur Oracle
- **Attention:**
  - On ne peut pas inclure SQLCODE ou SQLERRM directement dans une table.
  - On ne peut pas non plus utiliser SQLCODE ou SQLERRM dans une instruction SQL.

13/05/2005

Développement avec PL/SQL

16



## 7.4 Les fonctions propres à PL/SQL

```
DECLARE
    m_sqlcode NUMBER ;
    m_sqlerrm CHAR(70);
```

```
BEGIN
```

```
...
```

```
EXCEPTION
```

```
    WHEN OTHERS THEN
```

```
        m_sqlcode := SQLCODE ;
```

```
        m_sqlerrm := SQLERRM ;
```

```
        DBMS_OUTPUT.PUT_LINE('Il y a d''erreur - Code: '
                               ||m_sqlcode||' Message: ' ||m_sqlerrm )
```

```
END ;
```

Si aucune  
exception  
n'est active

SQLCODE = 0  
SQLERRM = 'normal,  
successful completion'

Si une  
exception  
définie par  
l'utilisateur

SQLCODE = 1  
SQLERRM = 'User-  
defined Exception'

13/05/2005

Développement avec PL/SQL

17

## 7.4 Les fonctions propres à PL/SQL

- Pour *affecter un numéro* à une exception définie par l'utilisateur

```
DECLARE
```

```
    PRAGMA EXCEPTION_INIT
```

```
        (<nom_exception_de_utilisateur>,<Num_erreur_Oracle>)
```

- **PRAGMA** est un mot-clé indiquant que l'instruction est une directive de compilateur

- **Num\_erreur\_Oracle**: est une valeur littérale, peut-être:

- Une erreur Oracle, telle que **-60**
- Une erreur située dans une plage allant de **-20000** à **-20999**

```
DECLARE
```

```
    contrainte_multi_tables EXCEPTION;
```

```
    PRAGMA EXCEPTION_INIT (contrainte_multi_tables, -60) ;
```

```
BEGIN
```

```
    RAISE contrainte_multi_tables;
```

```
END ;
```

13/05/2005

Développement avec PL/SQL

18

## 7.3 Les fonctions propres à PL/SQL

*Exemple 7.3:* 'Maximum 10 trajets par jour' et les autres exceptions?

```
DECLARE
trop_de_trajets EXCEPTION;
PRAGMA EXCEPTION_INIT
(trop_de_trajets, -60);
m_date DATE := '&Entrez_une_date';
nombre_de_trajets INTEGER;
m_sqlcode NUMBER ;
m_sqlerrm CHAR(70);

BEGIN
SELECT COUNT(*) INTO
nombre_de_trajets FROM trajet
WHERE datetrajet = m_date ;
IF nombre_de_trajets > 10 THEN
RAISE trop_de_trajets;
END IF ;
-- continuez et encore une erreur ....
m_date := '30-30-30';

EXCEPTION

WHEN trop_de_trajets THEN
DBMS_OUTPUT.PUT_LINE('Maximum
10 trajets par jour');
m_sqlcode := SQLCODE ;
m_sqlerrm := SQLERRM ;
DBMS_OUTPUT.PUT_LINE('Il y a
d''erreur - Code: '
||TO_CHAR(m_sqlcode));

WHEN OTHERS THEN
m_sqlcode := SQLCODE ;
m_sqlerrm := SQLERRM ;
DBMS_OUTPUT.PUT_LINE('Il y a
d''erreur - Code: '||TO_CHAR(m_sqlcode)
||' Message: ' ||m_sqlerrm );
END ;
```

13/05/2005

Développement avec PL/SQL

19

## 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

13/05/2005

Développement avec PL/SQL

20

## 8. Procédures et fonctions

- Introduction
- Procédures
- Fonctions

13/05/2005

Développement avec PL/SQL

21

### 8.1 Introduction

- Un sous-programme:
    - Est un bloc PL/SQL nommé qui peut accepter des paramètres et être invoqué d'un environnement appelant
    - Il y a deux types:
      - Une **procédure** qui exécute une action
      - Une **fonction** qui calcule une valeur
- Fournit la modularité, la réutilisabilité, et l'extensibilité.
- Fournit la maintenance facile; améliore la sécurité et l'intégrité des données, la performance et la clarté de code.

13/05/2005

Développement avec PL/SQL

22

## 8.1 Introduction

### ■ Structure de bloc pour sous-programmes PL/SQL:

**<header>**

**IS|AS**

*<section des déclararions>*

**BEGIN**

*<section exécution>*

**EXCEPTION**

*<section des exceptions>*

**END;**

La spécification du sous programme

Le corps du sous-programme

13/05/2005

Développement avec PL/SQL

23

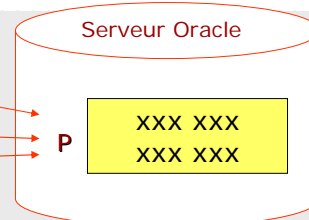
## 8.1 Introduction

### ■ Le serveur Oracle autorise:

- La compilation séparée des sous-programmes;
- Leurs stockage dans une base de données Oracle;
- Ils sont alors prêts à être exécutés.

```
P ;
xxxxxxxxxxxx
xxxxxxxxxxxx
P ;
xxxxxxxxxxxx
P ;
xxxxxxxxxxxx
xxxxxxxxxxxx
```

Le code s'est répété plus qu'une fois dans un programme PL/SQL



Le sous-programme P, qui contient le code répété

13/05/2005

Développement avec PL/SQL

24

## 8.2 Procédures

- Une **procédure** est un type de sous-programme qui exécute une action
- Une procédure peut être stockée dans la base de données, comme un objet de schéma, pour l'exécution répétée
- *Syntaxe pour création de procédures :*

```
CREATE [OR REPLACE] PROCEDURE nom_procédure
```

```
[( paramètre1 [mode1] type_donné1,  
  paramètre2 [mode2] type_donné2,  
  ... )]
```

```
IS | AS
```

```
bloc PL/SQL ;
```

13/05/2005

Développement avec PL/SQL

25

## 8.2 Procédures

- **Paramètre:** permet d'échanger dans les deux sens des informations entre le programme appelant et un sous-programme.
- **Mode:** Type d'argument:  
**IN** (défaut) | **OUT** | **IN OUT**
- **Type de donnée:** Le type de données de l'argument peut être n'importe quel type de données de SQL/PLSQL. Peut avoir de %TYPE, %ROWTYPE, ou n'importe quel type de données scalaire ou composé.
- **Bloc PL/SQL:** Le corps du sous-programme qui définit l'action exécutée selon la procédure

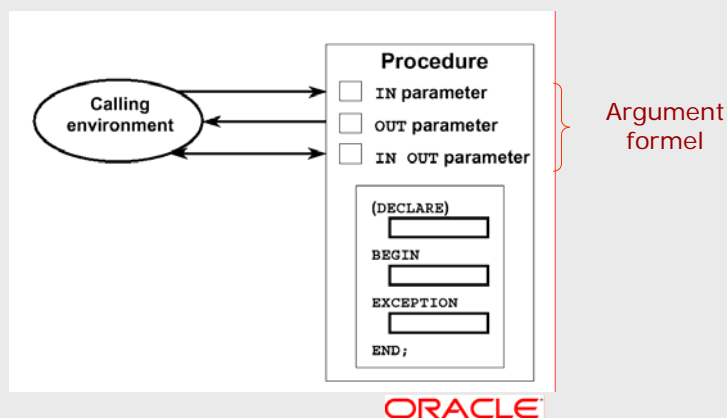
13/05/2005

Développement avec PL/SQL

26

## 8.2 Procédures

### ■ Les modes d'un paramètre



13/05/2005

Développement avec PL/SQL

27

## 8.2 Procédures

### ■ Les modes d'argument formel

IN	OUT	IN OUT
La valeur par défaut	Doit être spécifiée	Doit être spécifiée
Passe les valeurs au sous-programme	Retourne les valeurs à l'appel	Passe les valeurs initiales au sous-programme; retourne les valeurs mises à jour à l'appel
L'argument formel est traité comme une constante	L'argument formel agit comme une variable non-initialisée	L'argument formel agit comme une variable non-initialisée
On ne peut assigner une valeur à un argument formel	L'argument formel ne peut être utilisé dans cette expression: nécessite une valeur signée	Une valeur doit être assignée à l'argument formel
L'argument actuel peut être une constante, une variable initialisée, une chaîne de caractères ou une expression.	L'argument actuel doit être une variable	L'argument actuel doit être une variable

ORACLE

13/05/2005

Développement avec PL/SQL

28

## 8.2 Procédures

### ■ Exemple 8.1: Créer la procédure CREATE\_CH

```
CREATE OR REPLACE PROCEDURE create_ch
(p_nch      IN Ch.nch%TYPE ,
p_chauffeur IN Ch.chauffeur%TYPE)
IS
BEGIN
    INSERT INTO Ch VALUES (p_nch, p_chauffeur);
END ;
/
```

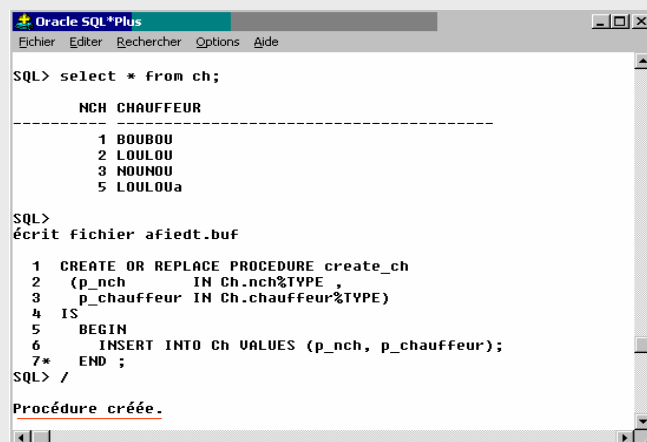
13/05/2005

Développement avec PL/SQL

29

## 8.2 Procédures

### ■ Exemple 8.1: Créer la procédure CREATE\_CH



```
Oracle SQL*Plus
Fichier  Edit  Rechercher  Options  Aide

SQL> select * from ch;

      NCH CHAUFFEUR
-----
1 BOUBOU
2 LOULOU
3 NOUMOU
5 LOULOUa

SQL>
écrit fichier afiedt.buf

1 CREATE OR REPLACE PROCEDURE create_ch
2 (p_nch      IN Ch.nch%TYPE ,
3 p_chauffeur IN Ch.chauffeur%TYPE)
4 IS
5 BEGIN
6     INSERT INTO Ch VALUES (p_nch, p_chauffeur);
7* END ;
SQL> /

Procédure créée.
```

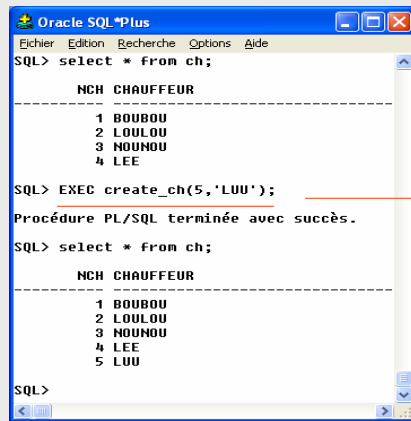
13/05/2005

Développement avec PL/SQL

30

## 8.2 Procédures

### ■ Exemple 8.1: Tester la procédure CREATE\_CH avec SQL\*Plus



```
Oracle SQL*Plus
Fichier Edition Recherche Options Aide
SQL> select * from ch;

      NCH CHAUFFEUR
-----
1 BOUBOU
2 LOULOU
3 NOUNOU
4 LEE

SQL> EXEC create_ch(5, 'LUU');

Procédure PL/SQL terminée avec succès.

SQL> select * from ch;

      NCH CHAUFFEUR
-----
1 BOUBOU
2 LOULOU
3 NOUNOU
4 LEE
5 LUU

SQL>
```

EXEC <commande\_PLSQL>

→ Exécuter une commande simple PL/SQL

L'environnement appelant

13/05/2005

Développement avec PL/SQL

31

## 8.2 Procédures

### ■ Exemple 8.1: Utiliser la procédure CREATE\_CH

```
DECLARE
    m_nch    Ch.nch%TYPE ;
    m_chauffeur Ch.chauffeur%TYPE
;
BEGIN
    m_nch := &numero_chauffeur;
    m_chauffeur := '&nom_chauffeur';
    create_ch(m_nch, m_chauffeur);
-- encore ...
END ;
```

L'environnement appelant

```
CREATE OR REPLACE PROCEDURE
create_ch
(p_nch    IN Ch.nch%TYPE ,
p_chauffeur IN
Ch.chauffeur%TYPE)
IS
BEGIN
    INSERT INTO Ch VALUES
(p_nch, p_chauffeur);
END ;
```

Le sous programme

13/05/2005

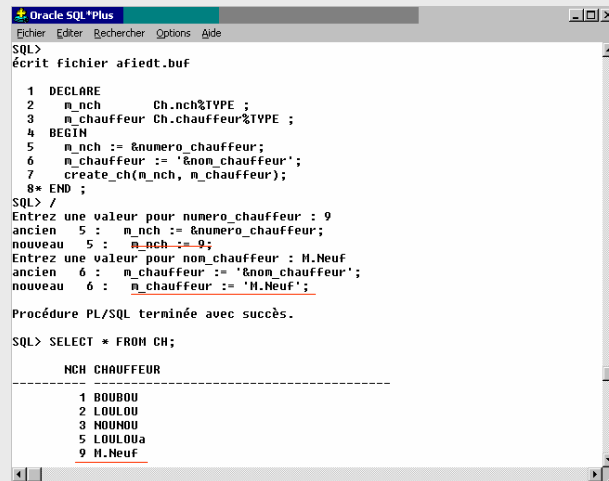
Développement avec PL/SQL

32



## 8.2 Procédures

### ■ Exemple 8.1: Utiliser la procédure CREATE\_CH



```
Oracle SQL*Plus
Echier  Edit  Rechercher  Options  Aide
SQL>
écrit fichier afiedt.buf

1 DECLARE
2   m_nch          Ch.nch%TYPE ;
3   m_chauffeur    Ch.chauffeur%TYPE ;
4 BEGIN
5   m_nch := &numero_chauffeur;
6   m_chauffeur := '&nom_chauffeur';
7   create_ch(m_nch, m_chauffeur);
8* END ;
SQL> /
Entrez une valeur pour numero_chauffeur : 9
ancien 5 : m_nch := &numero_chauffeur;
nouveau 5 : m_nch := 9;
Entrez une valeur pour nom_chauffeur : M.Neuf
ancien 6 : m_chauffeur := '&nom_chauffeur';
nouveau 6 : m_chauffeur := 'M.Neuf';

Procédure PL/SQL terminée avec succès.

SQL> SELECT * FROM CH;

      NCH CHAUFFEUR
-----
1 BOUBOU
2 LOULOU
3 NOUOU
5 LOULOUE
9 M.Neuf
```

13/05/2005

Développement avec PL/SQL

33

## 8.2 Procédures

### ■ Exemple 8.2: Créer la procédure GET\_NAME\_CH

```
CREATE OR REPLACE PROCEDURE get_name_ch
( p_nch    IN Ch.nch%TYPE ,
  p_chauffeur OUT Ch.chauffeur%TYPE )
IS
BEGIN
    SELECT chauffeur INTO p_chauffeur
    FROM Ch
    WHERE p_nch = Ch.nch;
END ;
/
```

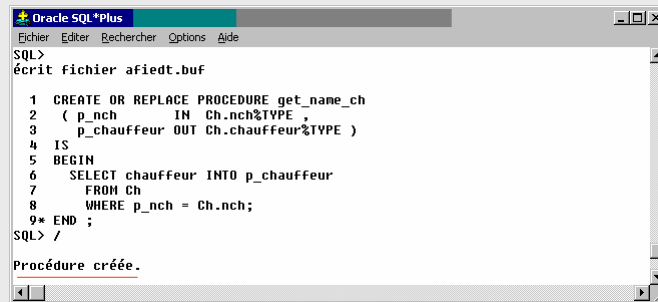
13/05/2005

Développement avec PL/SQL

34

## 8.2 Procédures

### ■ Exemple 8.2: Créer la procédure GET\_NAME\_CH



```
Oracle SQL*Plus
Echier  Edit  Rechercher  Options  Aide
SQL>
écrit fichier afiedt.buf
1 CREATE OR REPLACE PROCEDURE get_name_ch
2 ( p_nch      IN  Ch.nch%TYPE ,
3   p_chauffeur OUT Ch.chauffeur%TYPE )
4 IS
5 BEGIN
6   SELECT chauffeur INTO p_chauffeur
7     FROM Ch
8    WHERE p_nch = Ch.nch;
9* END ;
SQL> /
Procédure créée.
```

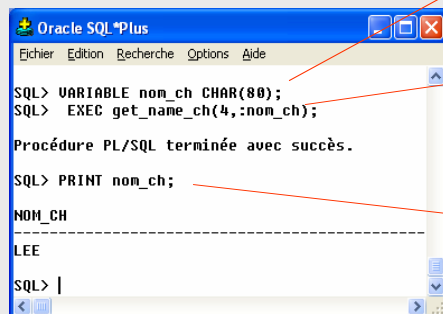
13/05/2005

Développement avec PL/SQL

35

## 8.2 Procédures

### ■ Exemple 8.1: Tester la procédure GET\_NAME\_CH avec SQL\*Plus



```
Oracle SQL*Plus
Echier  Edition  Recherche  Options  Aide
SQL> VARIABLE nom_ch CHAR(80);
SQL> EXEC get_name_ch(4,:nom_ch);
Procédure PL/SQL terminée avec succès.
SQL> PRINT nom_ch;
NOM_CH
-----
LEE
SQL> |
```

#### **VARIABLE <nom\_variable>**

→ Déclarer une variable de SQL\*Plus qui peut alors être faite référence dans PL/SQL

→ Référencer des variables en tapant un deux-points (:) suivi immédiatement par le nom de la variable

#### **PRINT <nom\_variable>**

→ Afficher la valeur actuelle d'une variable

L'environnement appelant

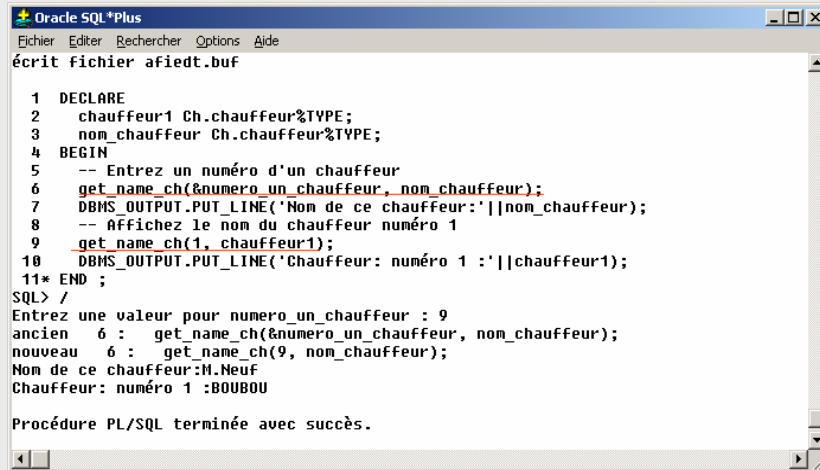
13/05/2005

Développement avec PL/SQL

36

## 8.2 Procédures

### ■ Exemple 8.2: Utiliser la procédure GET\_NAME\_CH



```
Oracle SQL*Plus
Fichier  Editer  Rechercher  Options  Aide
écrit fichier afiedt.buf

1 DECLARE
2   chauffeur1 Ch.chauffeur%TYPE;
3   nom_chauffeur Ch.chauffeur%TYPE;
4 BEGIN
5   -- Entrez un numéro d'un chauffeur
6   get_name_ch(&numero_un_chauffeur, nom_chauffeur);
7   DBMS_OUTPUT.PUT_LINE('Nom de ce chauffeur: '||nom_chauffeur);
8   -- Affichez le nom du chauffeur numéro 1
9   get_name_ch(1, chauffeur1);
10  DBMS_OUTPUT.PUT_LINE('Chauffeur: numéro 1 : '||chauffeur1);
11* END ;
SQL> /
Entrez une valeur pour numero_un_chauffeur : 9
ancien 6 : get_name_ch(&numero_un_chauffeur, nom_chauffeur);
nouveau 6 : get_name_ch(9, nom_chauffeur);
Nom de ce chauffeur:M.Neuf
Chauffeur: numéro 1 :BOUBOU

Procédure PL/SQL terminée avec succès.
```

13/05/2005

Développement avec PL/SQL

37

## 8.3 Fonctions

- Une fonction est un sous-programme qui calcule une valeur
- Fonctions et procédures offrent une structure similaire
- Mais une fonction contient une clause RETURN et une instruction RETURN

### ■ Syntaxe pour création de fonctions:

```
CREATE [OR REPLACE]
PROCEDURE nom_fonction
[( paramètre1 [mode1] type_donné1,
... )]
RETURN type_donné
IS | AS
[déclaration locale]
BEGIN
.....
l'instruction RETURN
END;
```

13/05/2005

Développement avec PL/SQL

38

## 8.3 Fonctions

■ *Continuez avec la procédure GET\_NAME\_CH ...*

```
Oracle SQL*Plus
Fichier  Edit  Rechercher  Options  Aide
SQL>
écrit fichier afiedt.buf

1 DECLARE
2   nom_chauffeur Ch.chauffeur%TYPE;
3 BEGIN
4   -- Entrez un numéro d'un chauffeur
5   get_name_ch(&numero_un_chauffeur, nom_chauffeur);
6   DBMS_OUTPUT.PUT_LINE('Nom de ce chauffeur:'||nom_chauffeur);
7* END ;
SQL> /
Entrez une valeur pour numero_un_chauffeur : 0
ancien 5 : get_name_ch(&numero_un_chauffeur, nom_chauffeur);
nouveau 5 : get_name_ch(0, nom_chauffeur);
DECLARE
*
ERREUR à la ligne 1 :
ORA-01403: Aucune donnée trouvée
ORA-06512: à "SYSTEM.GET_NAME_CH", ligne 6
ORA-06512: à ligne 5
```

13/05/2005

Développement avec PL/SQL

39

## 8.3 Fonctions

■ **Exemple 8.3: Créer la fonction GET\_NAME\_CH**

*S'il y a une erreur, retourne 0. Si non, retourne 1*

```
CREATE OR REPLACE FUNCTION
get_name_ch
( p_nch IN Ch.nch%TYPE ,
  p_chauffeur OUT
  Ch.chauffeur%TYPE )
RETURN INTEGER
IS
  m_count INTEGER ;
  m_result INTEGER ;
BEGIN
  -- Ce chauffeur existe déjà ?
  SELECT COUNT(*) INTO m_count
  FROM Ch
  WHERE p_nch = Ch.nch;

  IF m_count = 1 THEN
    SELECT chauffeur
    INTO p_chauffeur
    FROM Ch
    WHERE p_nch = Ch.nch;
    m_result := 1 ;
  ELSE
    m_result := 0 ;
  END IF ;
  RETURN m_result;
END ;
/
```

13/05/2005

Développement avec PL/SQL

40

## 8.3 Fonctions

### ■ Exemple 8.3: Créer la fonction GET\_NAME\_CH

```
Oracle SQL*Plus
Fichier  Edit  Rechercher  Options  Aide
SQL>
1 CREATE OR REPLACE FUNCTION get_name_ch
2   ( p_nch      IN  Ch.nch%TYPE ,
3     p_chauffeur OUT Ch.chauffeur%TYPE )
4   RETURN INTEGER
5   IS
6     m_count INTEGER ;
7     m_result INTEGER ;
8   BEGIN
9     -- Ce chauffeur existe déjà ?
10    SELECT COUNT(*) INTO m_count
11      FROM Ch
12     WHERE p_nch = Ch.nch;
13    IF m_count = 1 THEN
14      SELECT chauffeur INTO p_chauffeur
15        FROM Ch
16       WHERE p_nch = Ch.nch;
17      m_result := 1 ;
18    ELSE
19      m_result := 0 ;
20    END IF ;
21    RETURN m_result;
22* END ;
SQL> /
Fonction créée.
```

13/05/2005

Développement avec PL/SQL

41

## 8.3 Fonctions

### ■ Exemple 8.3: Utiliser la fonction GET\_NAME\_CH

```
Oracle SQL*Plus
Fichier  Edit  Rechercher  Options  Aide
SQL>
écrit fichier afiedt.buf
1 DECLARE
2   nom_chauffeur Ch.chauffeur%TYPE;
3 BEGIN
4   IF get_name_ch(&numero_un_chauffeur, nom_chauffeur) = 1 THEN
5     DBMS_OUTPUT.PUT_LINE('Nom de ce chauffeur: '||nom_chauffeur);
6   ELSE
7     DBMS_OUTPUT.PUT_LINE('Il n''y a pas ce chauffeur');
8   END IF ;
9* END ;
SQL> /
Entrez une valeur pour numero_un_chauffeur : 0
ancien 4 : IF get_name_ch(&numero_un_chauffeur, nom_chauffeur) = 1 THEN
nouveau 4 : IF get_name_ch(0, nom_chauffeur) = 1 THEN
Il n'y a pas ce chauffeur
Procédure PL/SQL terminée avec succès.
```

13/05/2005

Développement avec PL/SQL

42

## 8.3 Fonctions

- Supprimer les procédures:  
`DROP PROCEDURE nom_procédure ;`
- Supprimer les fonctions:  
`DROP FUNCTION nom_fonction ;`

13/05/2005

Développement avec PL/SQL

43

## Question 1 - TP.4

- Ecrivez un bloc PL/SQL pour:
  - Entrer le nom d'une équipe;
  - Afficher le budget, le nombre d'équipages et le nom du pays de cette équipe.
  - S'il y a une erreur, affichez un message: *'Ce nom n'existe pas'*

Conseil: Utiliser l'exception `NO_DATA_FOUND`

13/05/2005

Développement avec PL/SQL

44

## Question 1 - TP.4

```
DECLARE
m_equipe Equipe%ROWTYPE;
BEGIN
-- Entrer le nom d'une équipe
-- Sélectionnez les informations de cette équipe
-- Affichez les informations

EXCEPTION
WHEN NO_DATA_FOUND THEN
    → Il n'y a pas cette équipe;
END ;
```

13/05/2005

Développement avec PL/SQL

45

## Question 2 - TP.4

- Ecrivez une procédure pour ajouter un concurrent:
  - Entrer des informations d'un nouveau concurrent (c.à.d son nom, son prénom, son adresse, son nombre de participations, son pays ainsi que son numéro d'équipage);
  - Valider ces informations :
    - Si son numéro d'équipage n'existe pas dans la table EQUIPAGE, lancer une exception et afficher un message d'erreur;
    - Si son pays n'existe pas dans la table PAYS, lancer aussi une exception et afficher un message d'erreur;
  - Si les informations sont valides, insérer ce nouveau concurrent dans la table CONCURRENT.
  - Continuer de valider la règle : «Un équipage est composé de 2 concurrents. Lancer une exception si ce concurrent transgresse cette règle ». Si ce concurrent transgresse cette règle, lancer une exception, afficher un message d'erreur, et annuler la transaction effectuée.

Conseil: Utiliser les exceptions définies par l'utilisateur

13/05/2005

Développement avec PL/SQL

46

## Question 2 - TP.4

```
CREATE OR REPLACE PROCEDURE
create_concurrent
(p_concurrent IN
  Concurrent%ROWTYPE)
IS
  e equipages
    EXCEPTION ;
  e pays
    EXCEPTION ;
  e concurrents
    EXCEPTION ;
  m_count equipages  INTEGER
  ;
  m_count pays      INTEGER ;
  m_count concurrents INTEGER ;

BEGIN
  -- Validez NoEquipage
  -- Validez NomPays
  -- Ajoutez ce nouveau concurrent
  -- Validez RI : chaque équipage a
    maximum 2 concurrents
EXCEPTION
  WHEN e equipages THEN
    -- Erreur sur NoEquipages;
  WHEN e pays THEN
    -- Erreur sur NomPays;
  WHEN e concurrents THEN
    -- Le nombre de concurrents
    est égal ou moins que 2;
    ROLLBACK;
END ;
```

13/05/2005

Développement avec PL/SQL

47

## Question 3 - TP.4

- Ecrivez une procédure pour valider la règle :  
«Chaque pays a au moins un et maximum trois équipes»:
- → Afficher la liste de toutes les équipes qui transgressent cette règle (incluant le nom d'équipe, le nombre d'équipages, et les sponsors)

13/05/2005

Développement avec PL/SQL

48



### Question 3 - TP.4

```
DECLARE
CURSOR cur_pays
CURSOR cur_equipes
(p_NomPays)
BEGIN
-- Le titre du rapport
-- Traiter les pays
FOR rec_pays IN cur_pays
LOOP
-- Traiter 1 pays
-- Valider les règles

-- Erreur sur le nombre
d'Equipes
-- Afficher le nom de pays
-- Traiter les Equipes
FOR rec_equipes IN
cur_equipes
(rec_pays.NomPays)
LOOP
-- Traiter 1 Equipe
END LOOP ; -- les equipes
END LOOP ; -- les pays
-- La fin du rapport
END ;
```

13/05/2005

Développement avec PL/SQL

49

### Question 4a - TP.4

- a) Ecrivez une fonction pour traduire un entier (maximum 3 chiffres) à une chaîne de caractères.

Exemple: traduire\_entier(223) → 'Deux cents vingt trois'

*traduire\_chiffre(2) → 'deux'*

*traduire\_entier(223) → 'Deux cents vingt trois'*

13/05/2005

Développement avec PL/SQL

50

### Question 4a - TP.4

```
CREATE OR REPLACE FUNCTION traduction_chiffre
( p_number IN INTEGER)
-- Traduire un entier (1 chiffre)
RETURN VARCHAR
IS
  m_char VARCHAR(10) ;
BEGIN
  IF p_number = 1 THEN
    m_char := 'un';
  ELSIF p_number = 2 THEN
    ....
  ELSE -- p_number = 0
    m_char := '';
  END IF ;
  RETURN m_char;
END ;
```

13/05/2005

Développement avec PL/SQL

51

### Question 4a - TP.4

```
CREATE OR REPLACE FUNCTION traduction_entier
( p_number IN INTEGER)
-- Traduire un entier (3 chiffres)
RETURN VARCHAR
IS
  m_centaine NUMBER(1);
  m_dizaine  NUMBER(1);
  m_unite    NUMBER(1);
  m_char VARCHAR(200):= '' ;
BEGIN
  -- Calculer les centaine, dizaine, unité
  -- Traiter CENTAINE (cent /cents)
  -- Traiter DIZAINE et UNITE
  -- Dizaine <> 1 : vingt, trente, quarante, ...
  -- '-et-' ou non
  -- Dizaine = 1 :dix, onze, douze, ...
  RETURN TRIM(m_char);
END ;
```

### Question 4b - TP.4

b) Ecrivez une fonction pour traduire une somme à une chaîne de caractères.

Exemple: *traduire\_somme(1223.34) → 'Mille deux cents vingt trois francs vingt cinq centimes'*

13/05/2005

Développement avec PL/SQL

53

### Question 4b - TP.4

```
CREATE OR REPLACE FUNCTION traduction_somme
(p_somme IN real)
-- Traduire une somme
RETURN VARCHAR
IS
  m_billion NUMBER(3);
  m_million NUMBER(3);
  m_mille NUMBER(3);
  m_franc NUMBER(3);
  m_centime NUMBER(3);
  m_char VARCHAR(200) := " ";
BEGIN
  -- Calculer: m_centime, m_franc, m_mille, m_million, m_billion
  -- Traiter BILLION
  -- Traiter MILLION
  -- Traiter MILLE
  -- Traiter UNITE: 'un franc', '...francs';
  -- Traiter CENTIME
  RETURN m_char;
END ;
```