

Développement avec PL/SQL (1)

SYSINF-SES-CUI
UNIVERSITE DE GENEVE

Thang LE DINH
Email: 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 (items 1-4)
Partie 2 (items 5-6)
Partie 3 (items 7-8)

Références

- **PL/SQL User's Guide and Reference**
 - Oracle corporation
 - http://cui.unige.ch/DOC/oracle8/DOC/server803/A54654_01/title.htm
- **Oracle PL/SQL programming**
 - *Steven Feuerstein* – O'Reilly & Associates, Inc
- **Oracle8i: The Complete Reference**
 - *Kevin Loney, George Koch*
- **Oracle PL/SQL Interactive workbook**
 - *Benjamin Rosenzweig, Elena Silvestrova*
- **The complete Reference SQL**
 - *James R Groff and Paul N. Weinberg* – McGrawHill 2002
- **SQL pour Oracle**
 - *C.Souton, O.Teste* – Eyrolles 2004



30/03/2004

Développement avec PL/SQL

3

1. Introduction

- Définition
- Les différences entre PL/SQL et SQL
- Les éléments d'un bloc PL/SQL
- Les moteurs PL/SQL
- Pratique



30/03/2004

Développement avec PL/SQL

4

1.1 Définition

- PL/SQL: un langage procédural de traitement transactionnel manipulant des tables d'une base de données Oracle

PL/SQL =

Les commandes
SQL

+

Les structures et instructions
dans les langages de
programmation
traditionnels

(ex: boucles, branchements,
affectations et déclaration)



30/03/2004

Développement avec PL/SQL

5

1.2 Les différences entre PL/SQL et SQL

SQL

PL/SQL

Un langage non procédural	Un langage procédural: <i>Permettre de déterminer aussi bien la tâche à exécuter que la manière de l'exécuter</i>
Les commandes sont traitées une par une	Permettre de regrouper logiquement un ensemble de commandes SQL en un seul bloc
	Un bloc PL/SQL: -Les commandes SQL -Les commandes de traitement transactionnel -Branchement; Répétition; Affectation



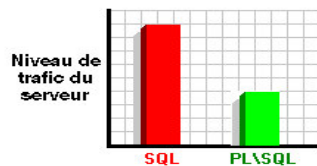
30/03/2004

Développement avec PL/SQL

6

1.2 Les différences entre PL/SQL et SQL

SQL	PL/SQL
	Gestion des exceptions pour le traitement des erreurs
Chaque instruction SQL provoque un appel vers le serveur	PL/SQL envoyant un bloc entier d'instructions SQL au serveur en une seule fois → Une amélioration de l'ensemble des performances

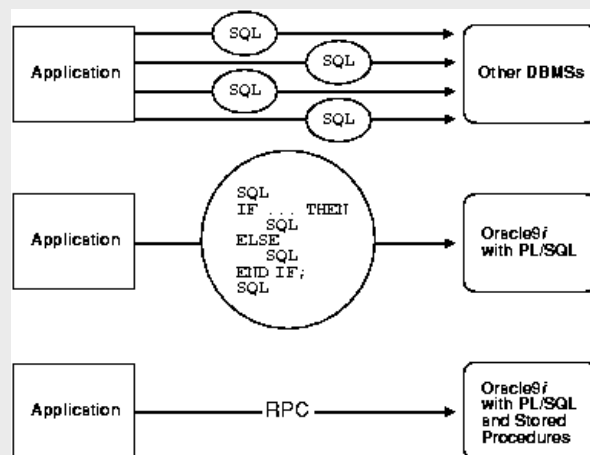


30/03/2004

Développement avec PL/SQL

7

1.3 Les moteurs PL/SQL

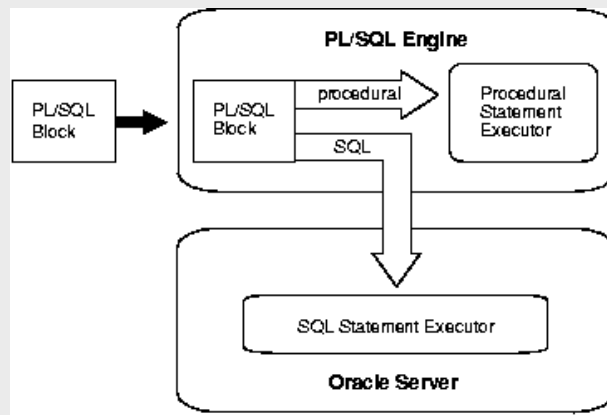


30/03/2004

Développement avec PL/SQL

8

1.3 Les moteurs PL/SQL

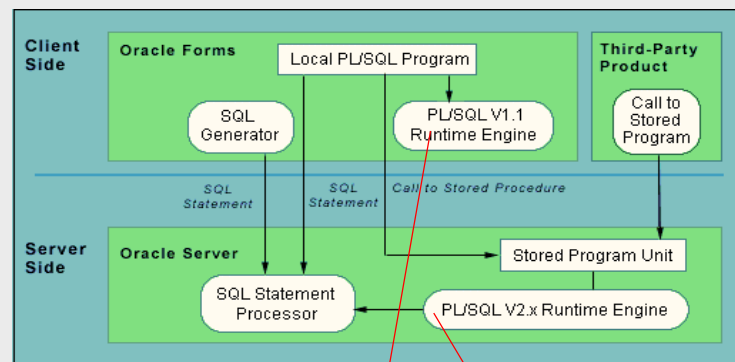


30/03/2004

Développement avec PL/SQL

9

1.3 Les moteurs PL/SQL



Un moteur de côté de client qui permet à l'application d'exécuter local des programmes

Un moteur de côté de serveur qui exécute des programmes stockés



30/03/2004

Développement avec PL/SQL

10

1.4 Les éléments d'un bloc PL/SQL

Une partie déclarative (facultative):

→ définitions des variables, constantes, enregistrements, curseurs et exceptions

Une partie exécutable (obligatoire):

→ pour les commandes exécutables et les procédures standards

Une partie pour les traitements des erreurs (facultative):

→ pour les routines spécifiant le traitement approprié dès qu'une erreur survient

DECLARE

<section des déclararions>

BEGIN

<section exécution>

EXCEPTION

<section des exceptions>

END;



30/03/2004

Développement avec PL/SQL

11

1.4 Les éléments d'un bloc PL/SQL

Exemple 1.1:

BEGIN

/ Manipulation des tuples de la table chauffeur*

*CH (NCH, CHAUFFEUR) */*

-- Insérer un nouveau tuple

INSERT INTO ch VALUES (5, 'Mr XXX');

-- Modifier un tuple

UPDATE ch SET chauffeur = 'Mr YYY' WHERE nch = 5;

-- Supprimer ce tuple

DELETE FROM ch WHERE nch = 5 ;

END ;

/



30/03/2004

Développement avec PL/SQL

12

1.4 Les éléments d'un bloc PL/SQL

■ Quelques règles de syntaxe:

■ *Terminaison:*

- Les instructions doivent se terminer par un point-virgule (;)

■ *Les commentaires sur une seule ligne:*

- Les commentaires doivent être précédés de deux tirets (--)

■ *Les commentaires multi-lignes:*

- Encadrés d'une combinaison barre oblique et astérisque
(/* */)



30/03/2004

Développement avec PL/SQL

13

1.5 Pratique: un bloc PL/SQL

■ **Sujet: GESTION D'UN PARC DE VEHICULES (PARCVEH)**

- **VOITURE** (NOV, MV, KM, PSG)
- **CH** (NCH, CHAUFFEUR)
- **V_CH** (NOV, NCH, NKM)
- **REPARATION** (NOREP, NOV, NOG, TYPREP, PX, KMCPT)
- **TRAJET** (NOTRAJ, VILLEDEP, VILLEARR, DATETRAJET, NBKM)
- **TR_NOV** (NOTRAJ, NOV, NCH, NBPERSTR)



30/03/2004

Développement avec PL/SQL

14

1.5 Pratique: un bloc PL/SQL

■ **Sujet: GESTION D'UN PARC DE VEHICULES (PARCVEH)**

- **VOITURE (NOV, MV, KM, PSG):** à une voiture on associe son numéro de voiture NOV qui la distingue des autres voitures, sa marque MV, le nombre de kilomètres qu'elle a parcourus KM, le nombre de places disponibles de passagers PSG.
- **CH (NCH, CHAUFFEUR):** à un numéro de chauffeur NCH on associe un seul nom du chauffeur CHAUFFEUR.
- **V-CH (NOV, NCH, NKM):** le chauffeur de tel numéro NCH a conduit la voiture de tel numéro NOV pendant tant de kilomètres NKM depuis que la voiture est en service.



30/03/2004

Développement avec PL/SQL

15

1.5 Pratique: un bloc PL/SQL

- **REPARATION (NOREP, NOV, NOG, TYPREP, PX, KMCPT):** la voiture de tel numéro NOV est menée au garage de tel numéro NOG pour une réparation de numéro NOREP et de type TYPERP; elle a alors tant de kilomètres au compteur KMCPT. Cette réparation a coûté tant PX.
- **TRAJET (NOTRAJ, VILLEDEP, VILLEARR, DATE_TRAJET, NBKM):** un trajet de tel numéro NOTRAJ a été effectué à telle date DATE-TRAJET; les villes de départ et d'arrivée sont respectivement VILLEDEP, VILLEARR; le trajet est de tant de kilomètres NBKM.
- **TR-NOV (NOTRAJ, NOV, NCH, NBPERSTR):** la voiture de numéro NOV, conduite par le chauffeur de numéro NCH, a transporté tant de personnes (NBPERSTR) pour le trajet de numéro NOTRAJ.



30/03/2004

Développement avec PL/SQL

16

1.5 Pratique: un bloc PL/SQL

■ Exemple 1.1: First_block

```
BEGIN
  /* Manipulation des tuples de la table chauffeur
    CH (NCH, CHAUFFEUR) */

  -- Insérer un nouveau tuple
  INSERT INTO ch VALUES (5, 'Mr XXX');

  -- Modifier un tuple
  UPDATE ch SET chauffeur = 'Mr YYY' WHERE nch = 5;

  -- Supprimer ce tuple
  DELETE FROM ch WHERE nch = 5 ;
END ;
```



30/03/2004

Développement avec PL/SQL

17

1.5 Pratique: un bloc PL/SQL

- **Créer un bloc PL/SQL**
 - Lancer SQL*Plus
 - Lancer l'éditeur de texte
 - Sauvegarder un bloc
- **Exécuter un bloc**



30/03/2004

Développement avec PL/SQL

18

1.5 Pratique: un bloc PL/SQL

■ Lancer SQL*Plus



Connexion avec
SQL*Plus pour
Windows

```
Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 9.0.1.0.1 - Production on Thu Apr 11 15:35:59 2002
(c) Copyright 2001 Oracle Corporation. All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.0.1.1.1 - Production
With the Partitioning option
JServer Release 9.0.1.1.1 - Production

SQL> |
```

Connexion à distance

```
>telnet cuisuna.unige.ch
>source /unige/cui/env/oracle.csh
>sqlplus
```



30/03/2004

Développement avec PL/SQL

19

1.5 Pratique: un bloc PL/SQL

■ Lancer l'éditeur de texte

Menu: Editeur – Editeur – Appeler éditeur

```
afiedt.buf - Notepad
File Edit Format Help

BEGIN
/* Manipulation des tuples de la table chauffeur
CH (NCH, CHAUFFEUR) */
-- Insérer un nouveau tuple
INSERT INTO ch VALUES (5, 'Mr XXX');

-- Modifier un tuple
UPDATE ch SET chauffeur = 'Mr YYY' WHERE nch = 5;

-- Supprimer ce tuple
DELETE FROM ch WHERE nch = 5 ;

END ;
```



30/03/2004

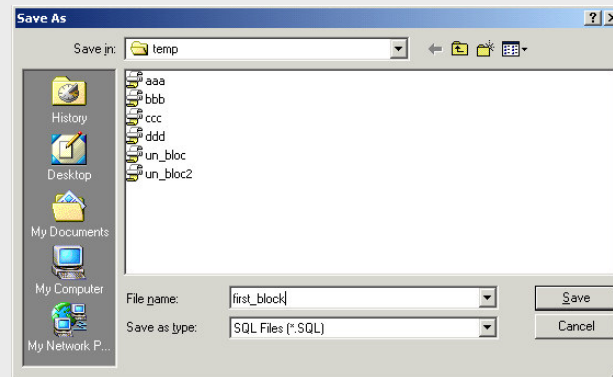
Développement avec PL/SQL

20

1.5 Pratique: un bloc PL/SQL

■ Sauvegarder un bloc / un script

Menu: Fichier – Sauvegarder sous ...



30/03/2004

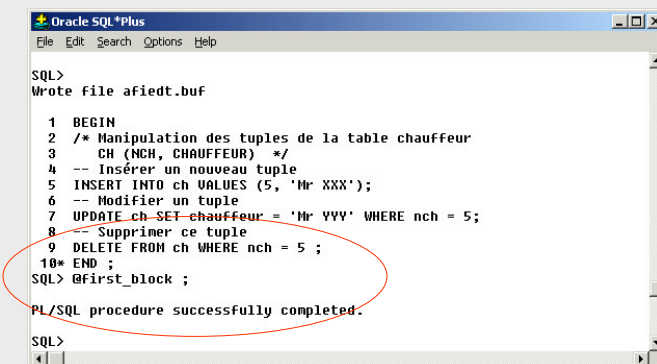
Développement avec PL/SQL

21

1.5 Pratique: un bloc PL/SQL

■ Exécuter un bloc

SQL> @<nom_bloc> ;



30/03/2004

Développement avec PL/SQL

22

1.5 Pratique: un bloc PL/SQL

■ Command-line:

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> BEGIN
2  /* Manipulation des tuples de la table chauffeur
3  CH (NCH, CHAUFFEUR) */
4  -- Insérer un nouveau tuple
5  INSERT INTO ch VALUES (5, 'Mr XXX');
6  -- Modifier un tuple
7  UPDATE ch SET chauffeur = 'Mr VVV' WHERE nch = 5;
8  -- Supprimer ce tuple
9  DELETE FROM ch WHERE nch = 5 ;
10 END ;
11 .
SQL> /

PL/SQL procedure successfully completed.

SQL> SAVE second_block ;
Created file second_block.sql
SQL> @second_block ;

PL/SQL procedure successfully completed.

SQL>
```

Entrer un bloc

Terminer un bloc

Exécuter un bloc (à partir de buffer-SQL)

Sauvegarder un bloc

Exécuter un bloc (à partir d'un fichier .SQL)



30/03/2004

Développement avec PL/SQL

23

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 (items 2-4)
Partie 2 (items 5-6)
Partie 3 (items 7-8)



30/03/2004

Développement avec PL/SQL

24

2. Variables et constantes

- Description des variables et constantes
- Type de données
- Déclarations des variables et constantes
- Affectation de valeur
- Pratique



30/03/2004

Développement avec PL/SQL

25

2.1 Description des variables et constantes

- **Définitions:**
 - *Une constante*: sa valeur doit **rester la même** durant toute l'exécution du bloc;
 - *Une variable*: sa valeur **peut changer** durant l'exécution du bloc
- **But:**
 - **Stocker les valeurs** des données en vue d'un traitement ultérieur
 - **Calculer les valeurs** à insérer dans une instruction ou une table

DECLARE

< Section des déclarations >

*voici les déclarations
des variable et constantes*

BEGIN

END ;

/



30/03/2004

Développement avec PL/SQL

26

2.1 Description des variables et constantes

- Quelques règles pour les **noms** des variables et constantes:
 - De 1 à 30 caractères
 - Commencer par une **lettre**, les autres pouvant être des lettres, des chiffres ou les symboles spéciaux _ \$ #
 - Majuscules ou minuscules sont employées indifféremment
 - Le nom variable ne doit pas être le même comme le nom de *tables*, des *colonnes* et des *mots réservés*
→ Utiliser les *préfixe*



30/03/2004

Développement avec PL/SQL

27

2.2 Type de données

Scalaire: variables, constituées d'une seule valeur

TYPE	Description
NUMBER	NUMBER(précision, position) La précision est le nombre de chiffres La position représente le nombre de chiffres après la virgule
BINARY_INTEGER	Type de base pour entiers entre -2,147,483,647 et 2,147,483,647
INTEGER	Sous-type de NUMBER : NUMBER(38)
BOOLEAN	Stocker une des trois valeurs suivantes: TRUE , FALSE ou NULL
DATE	Une longueur fixe pour date et heure
CHAR	CHAR(longueur) Chaîne de caractères alpha-numériques de longueur fixe. Les valeurs admises vont de 1 à 32767 octets
VARCHAR	VARCHAR(longueur) Chaîne de caractères alphanumériques de longueur variable.



30/03/2004

Développement avec PL/SQL

28

2.2 Type de données

Composite: variables qui se composent de plusieurs valeurs

TYPE	Description
RECORD	Stocker les données sous forme d' <i>enregistrements</i>
TABLE	Stocker les données dans des <i>tables PL/SQL</i>



30/03/2004

Développement avec PL/SQL

29

2.3 Déclarations des variables et constantes

■ Déclarations des variables et constantes

■ Déclarations bornées

- Pour déclarer les variables scalaires
(Par exemple: chaîne de caractère, date, booléenne, entier)

■ Déclarations basées

- Déclarez une variable selon:
 - une définition de *colonne* de base de données, ou
 - une *autre variable* précédemment déclarée, ou
 - une définition de *table* de base de données



30/03/2004

Développement avec PL/SQL

30

2.3 Déclarations des variables et constantes

■ *Déclarations bornées:*

■ Déclaration d'une variable

Nom_variable *type_données* [NOT NULL]
[:= | DEFAULT valeur_initiale]

■ Déclaration d'une constante

Nom_constante *type_données*
CONSTANT := valeur_initiale



30/03/2004

Développement avec PL/SQL

31

2.3 Déclarations des variables et constantes

■ Exemple 2.1:

DECLARE

-- Déclarations des variables:

total_cde NUMBER(15,2) ;

adresse_client VARCHAR(40) ;

nombre_client BINARY_INTEGER ;

demain DATE **DEFAULT** SYSDATE+1 ;

client_valide BOOLEAN NOT NULL **:=** TRUE ;

-- Déclarations des constantes:

minimal_de_cde NUMBER(1) **CONSTANT :=** 5 ;

BEGIN

-- Faire quelques choses

END



30/03/2004

Développement avec PL/SQL

32

2.3 Déclarations des variables et constantes

■ Déclarations basées:

- Les objets de base de données possèdent certains attributs que vous pouvez utiliser pour simplifier les déclarations

■ L'attribut %TYPE

- renvoie le type de données et la taille de la variable ou la colonne

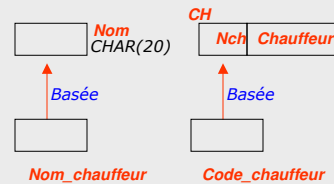
Nom_variable_basée **Nom_variable_origine%TYPE**

ou

Nom_variable_basée **Nom_table.Nom_colonne%TYPE**

DECLARE

```
nom          CHAR(20);
nom_chauffeur nom%TYPE;
code_chauffeur ch.nch%TYPE;
```



30/03/2004

Développement avec PL/SQL

33

2.3 Déclarations des variables et constantes

■ Déclarations basées:

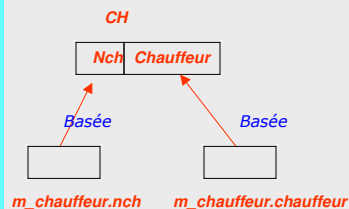
■ L'attribut %ROWTYPE

- donne à la variable la même structure (c.à.d noms de champ et types de données) que celle appliquée à une ligne dans une table

Nom_variable_basée **Nom_table%ROWTYPE**

DECLARE

```
m_chauffeur  ch%ROWTYPE;
BEGIN
m_chauffeur.nch := 5;
m_chauffeur.chauffeur := 'THOMAS';
END;
```



30/03/2004

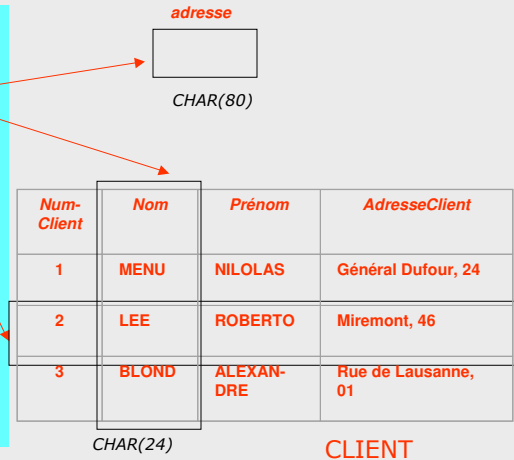
Développement avec PL/SQL

34

2.3 Déclarations des variables et constantes

■ Exemple 2.2:

```
DECLARE
adresse CHAR(80);
adresse_client adresse%TYPE;
nom_client client.nom%TYPE;
m_client client%ROWTYPE;
BEGIN
nom_client := 'ABC';
m_client.numclient := 1;
m_client.prénom := 'THOMAS';
-- Faire quelques choses
END;
```



30/03/2004

Développement avec PL/SQL

35

2.4 Affectation de valeur

■ Affectation de valeur

- Utiliser l'instruction déclarative de la variable
- Utiliser l'instruction PL/SQL
- Sélectionner des valeurs de base de données et affecter les à des variables



30/03/2004

Développement avec PL/SQL

36

2.4 Affectation de valeur

- Utiliser l'instruction déclarative de la variable

- Exemple 2.3:

```
DECLARE
-- Valeurs par défaut:
ordre      NUMBER (1) DEFAULT 1 ;
code_client CHAR(3) := 'ABC' ;
demain     DATE NOT NULL := '12-06-02' ;
possible   BOOLEAN := NULL ;
d_accord   BOOLEAN := TRUE ;
BEGIN
-- Faire quelques choses
END ;
```



30/03/2004

Développement avec PL/SQL

37

2.4 Affectation de valeur

- Utiliser l'instruction PL/SQL

`plsql_variable := plsql_expression`

⊘ Le type de données doit être le même à gauche et à droite de l'affectation

- Exemple 2.4:

```
DECLARE
ordre      NUMBER (1) DEFAULT 1 ;
code_client CHAR(3) := 'ABC' ;
demain     DATE := '12-06-02' ;
possible   BOOLEAN := NULL ;
d_accord   BOOLEAN := TRUE ;
BEGIN
ordre := 2 ;
code_client := code_client || 'BEF' ;
demain := SYSDATE + 1 ;
possible := NOT d_accord ;
END ;
```



30/03/2004

Développement avec PL/SQL

38

2.4 Affectation de valeur

- Sélectionner des valeurs de base de données et affecter les à *des variables*

SELECT les_attributs INTO les_variables FROM ...

- Exemple 2.5:

```
DECLARE
nombre_ch      INTEGER;
mv_nov_13      voiture.mv%TYPE;
km_nov_13      voiture.km%TYPE;

BEGIN

SELECT COUNT(nch) INTO nombre_ch FROM ch;

SELECT mv, km INTO mv_nov_13, km_nov_13
FROM voiture WHERE nov = '13';

END ;
```



30/03/2004

Développement avec PL/SQL

39

2.5 Pratique: Variables et constantes

- Exemple 2.6:

- Écrivez un bloc PL/SQL pour entrer un nouveau chauffeur

- Solution 1: Utiliser %TYPE

```
DECLARE
n_nch          ch.nch%TYPE;
n_chauffeur    ch.chauffeur%TYPE;

BEGIN

n_nch          := &Numero_chauffeur ;
n_chauffeur    := '&Nom_chauffeur' ;

INSERT INTO ch VALUES (n_nch, n_chauffeur);

END ;

/
```



30/03/2004

Développement avec PL/SQL

40

2.5 Pratique: Variables et constantes

- **&Numero_chauffeur** (numéro du chauffeur) définit une *variable de substitution* qui invite SQL*Plus à entrer une chaîne de données demandée à l'utilisateur

BEGIN

n_nch := &Numero_chauffeur ;

n_chauffeur :=
'&Nom_chauffeur' ;

INSERT INTO ch VALUES
(n_nch, n_chauffeur);

END ;

/

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> @exemple3 ;
Enter value for numero_chauffeur: 16
old 5: n_nch      := &Numero_chauffeur ;
new 5: n_nch      := 16 ;
Enter value for nom_chauffeur: THANG
old 6: n_chauffeur := '&Nom_chauffeur' ;
new 6: n_chauffeur := 'THANG' ;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM ch WHERE nch = 16 ;

      NCH CHAUFFEUR
-----
      16 THANG

SQL> |
```



30/03/2004

Développement avec PL/SQL

41

2.5 Pratique: Variables et constantes

- Exemple 2.7 - Solution 2:
 - Utiliser %ROWTYPE

DECLARE

n_ch ch%ROWTYPE;

BEGIN

n_ch.nch := &Numero_chauffeur ;

n_ch.chauffeur := '&Nom_chauffeur' ;

INSERT INTO ch VALUES (n_ch.nch, n_ch.chauffeur);

END ;

/



30/03/2004

Développement avec PL/SQL

42

2.5 Pratique: Variables et constantes

- Exemple 2.8:

- *Affichez le total moyen des nombres de kilomètres de tous les trajets pendant la période de 01.01.2001 à 30.06.2001*

- Solution:

```
DECLARE
total_moyen    NUMBER;
BEGIN
SELECT AVG(nbkkm) INTO total_moyen FROM trajet
WHERE datetrajet BETWEEN '01-01-01' AND '30-06-01' ;
DBMS_OUTPUT.PUT_LINE('Total moyen: '||TO_CHAR(total_moyen));
END ;
/
```



30/03/2004

Développement avec PL/SQL

43

2.5 Pratique: Variables et constantes

- Explication:

- **DBMS_OUTPUT.PUT_LINE()**

→ Afficher les variables et les expressions PL/SQL



DBMS_OUT_PUT.PUT_LINE(les_expressions)

→ Affichez *les_expressions* et

→ Insérez une nouvelle ligne

DBMS_OUT_PUT.PUT(les_expressions)

→ Affichez *les_expressions* dans même ligne

DBMS_OUT_PUT.NEW_LINE;

→ Insérez une nouvelle ligne



30/03/2004

Développement avec PL/SQL

44

2.5 Pratique: Variables et constantes

- De plus, il faut demander SQL*Plus de permettre d'afficher

- **SET SERVEROUTPUT ON | OFF**

ON: permettre d'afficher | OFF : ne permettre pas

```
Oracle SQL*Plus
File Edit Search Options Help
1 DECLARE
2   total_moyen      NUMBER;
3 BEGIN
4   SELECT AVG(nbk) INTO total_moyen
5   FROM trajet
6   WHERE datetrajet BETWEEN '01-01-01' AND '30-06-01' ;
7   DBMS_OUTPUT.PUT_LINE('Total moyen: ' || TO_CHAR(total_moyen));
8* END ;
SQL> SET SERVEROUTPUT ON ;
SQL> @exemple4 ;
Total moyen:

PL/SQL procedure successfully completed.
SQL> SET SERVEROUTPUT OFF ;
SQL>
```



30/03/2004

Développement avec PL/SQL

45

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 | | |



30/03/2004

Développement avec PL/SQL

46

3. Types de données

- Variable de type **table**
- Variable de type **record**



30/03/2004

Développement avec PL/SQL

47

3.1 Variable de type table

- Variable de type **table** : les tables PL/SQL:
 - Une table PL/SQL doit comporter **une clé primaire** et **une colonne**.
 - Une table PL/SQL grandit à mesure que sont **ajoutées** de nouvelles lignes;
 - La **valeur clé primaire** (PRIMARY KEY) agit comme **un pointeur** sur une valeur stockée dans la table PL/SQL;
 - La clé primaire doit relever du type de données **entiers binaires** (BINARY_INTEGER)

Clé primaire	Colonne
1	AAA
2	ZZZ
3	NNN
4	CCC
500	A-B-C



30/03/2004

Développement avec PL/SQL

48

3.1 Variable de type table

- Définir un type de TABLE

- Syntaxe:

```
TYPE nom_type  
IS TABLE OF  
{type_donnée | variable%TYPE  
 | table.colonne%TYPE}  
[NOT NULL]  
INDEX BY BINARY_INTEGER;
```

- Exemple:

Clé primaire	Colonne
0	ZERO
1	UN
2	DEUX
3	TROIS
4	QUATRE
...	...
9	NEUF

- Définir un type

```
TYPE caractere_chiffre_type IS TABLE OF CHAR(10) NOT NULL  
INDEX BY BINARY_INTEGER ;
```



30/03/2004

Développement avec PL/SQL

49

3.1 Variable de type table

- Déclarer les tables PL/SQL de ce type:

```
Nom_table_PLSQL Nom_type ;
```

- Référencer les lignes d'une table PL/SQL:

→ Spécifier une clé primaire

```
Nom_table_PLSQL (valeur_clé_primaire);
```

Exemple:

```
DECLARE  
TYPE caractere_chiffre_type IS TABLE OF CHAR(10) NOT NULL  
INDEX BY BINARY_INTEGER ;  
car_tab caractere_chiffre_type ;  
  
BEGIN  
car_tab (0) := 'ZERO' ;  
....  
END ;
```



30/03/2004

Développement avec PL/SQL

50

3.1 Variable de type table

■ Exemple 3.1: Entrer un chiffre et afficher ce chiffre en caractères

```
DECLARE
-- Chiffre à entrer
ce_chiffre INTEGER(1);
-- chaîne de caracteres à afficher
car_chiffre CHAR(10);
-- Définir un type
TYPE caractere_chiffre_type IS
TABLE OF CHAR(10) NOT NULL
INDEX BY BINARY_INTEGER;
-- Déclarer la table PL/SQL de ce type
car_tab caractere_chiffre_type;

BEGIN
-- Affectations
car_tab (0) := 'ZERO';
car_tab (1) := 'UN';

car_tab (2) := 'DEUX';
car_tab (3) := 'TROIS';
car_tab (4) := 'QUATRE';
car_tab (5) := 'CINQ';
car_tab (6) := 'SIX';
car_tab (7) := 'SEPT';
car_tab (8) := 'HUIT';
car_tab (9) := 'NEUF';

-- Entrer un chiffre
ce_chiffre := &ENTER_UN_CHIFFRE;
-- Afficher ce chiffre en caracteres
DBMS_OUTPUT.PUT_LINE('En
caracteres : ' || car_tab (ce_chiffre));
END;
```



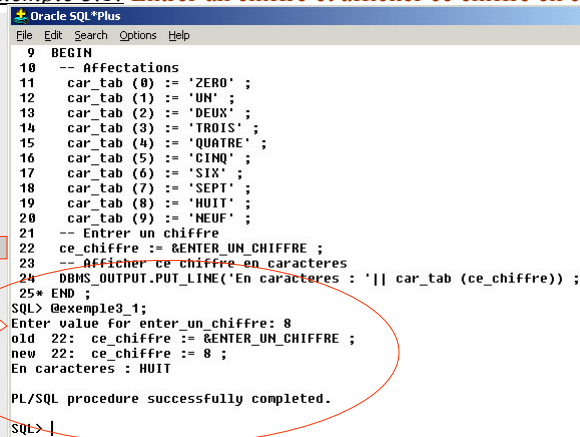
30/03/2004

Développement avec PL/SQL

51

3.1 Variable de type table

■ Exemple 3.1: Entrer un chiffre et afficher ce chiffre en caractères



```
9 BEGIN
10 -- Affectations
11 car_tab (0) := 'ZERO';
12 car_tab (1) := 'UN';
13 car_tab (2) := 'DEUX';
14 car_tab (3) := 'TROIS';
15 car_tab (4) := 'QUATRE';
16 car_tab (5) := 'CINQ';
17 car_tab (6) := 'SIX';
18 car_tab (7) := 'SEPT';
19 car_tab (8) := 'HUIT';
20 car_tab (9) := 'NEUF';
21 -- Entrer un chiffre
22 ce_chiffre := &ENTER_UN_CHIFFRE;
23 -- Afficher ce chiffre en caracteres
24 DBMS_OUTPUT.PUT_LINE('En caracteres : ' || car_tab (ce_chiffre));
25 END;
SQL> @exemple3_1;
Enter value for enter_un_chiffre: 8
old 22: ce_chiffre := &ENTER_UN_CHIFFRE;
new 22: ce_chiffre := 8;
En caracteres : HUIT

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



30/03/2004

Développement avec PL/SQL

52

3.2 Variable de type record

- Les **records** (**enregistrement**) PL/SQL sont *composés de champs* chacun un nom unique et pouvant appartenir à différents types de données

- Par exemple: **RECORD Chauffeur**

Code	Nom
INTEGER(3)	CHAR(40)

RECORD temps

Heure	Minute	Seconde
INTEGER(2)	INTEGER(2)	INTEGER(2)

- Le type RECORD permet de recueillir des informations sur les attributs d'un objet



30/03/2004

Développement avec PL/SQL

53

3.2 Variable de type record

- Définir un type de RECORD

```
TYPE nom_type_record IS RECORD
( nom_champ_1 {type_donnée | variable%TYPE |
table.colonne%TYPE} [NOT NULL]),
( nom_champ_2 {type_donnée | variable%TYPE |
table.colonne%TYPE} [NOT NULL]),
....
);
```

DECLARE

```
TYPE ch_type IS RECORD
(ch_code NUMBER(3),
ch_nom CHAR(40)); -- Définir un type

un_chauffeur ch_type ; -- Déclarer une variable
```



30/03/2004

Développement avec PL/SQL

54

3.2 Variable de type record

- Déclarer les enregistrements de ce type
`Nom_enregistrement Nom_type_record ;`
- Référencer individuellement les champs d'un enregistrement
`Nom_enregistrement.nom_champ;`

```
DECLARE
TYPE ch_type IS RECORD (ch_code NUMBER(3) ,
                        ch_nom CHAR(40)); -- Définir un type
un_chauffeur ch_type ; -- Déclarer une variable
BEGIN
    un_chauffeur.ch_code := &Numero_chauffeur ; -- Référencer les champs
    un_chauffeur.ch_nom := '&Nom_chauffeur' ;
END ;
```



30/03/2004

Développement avec PL/SQL

55

3.3 Pratique: Variable de type record

Exemple 3.2: Entrer un chauffeur

```
DECLARE
TYPE ch_type IS RECORD
(ch_code NUMBER(3) ,
 ch_nom CHAR(40)); -- Définir un type
un_chauffeur ch_type ; -- Déclarer une variable
BEGIN
    un_chauffeur.ch_code := &Numero_chauffeur ; -- Référencer les champs
    un_chauffeur.ch_nom := '&Nom_chauffeur' ;
    INSERT INTO ch
        VALUES (un_chauffeur.ch_code, un_chauffeur.ch_nom);
END ;
```



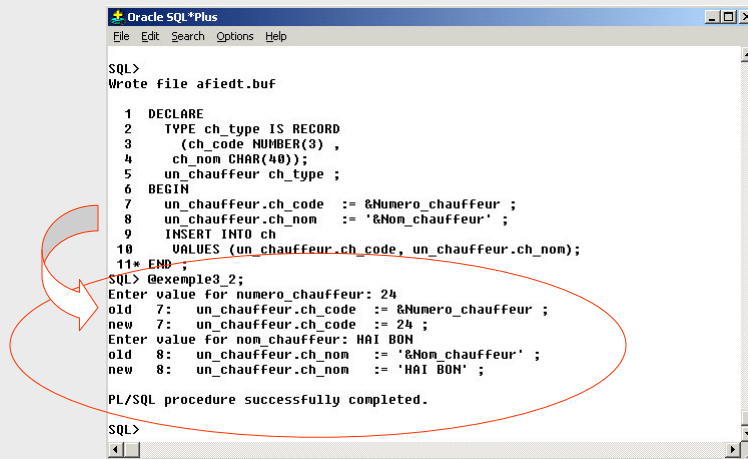
30/03/2004

Développement avec PL/SQL

56

3.3 Pratique: Variable de type record

Exemple 3.2: Entrer 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   un_chauffeur.ch_code := &Numero_chauffeur ;
 8   un_chauffeur.ch_nom  := '&Non_chauffeur' ;
 9   INSERT INTO ch
10     VALUES (un_chauffeur.ch_code, un_chauffeur.ch_nom);
11* END ;
SQL> @exemple3_2;
Enter value for numero_chauffeur: 24
old 7:   un_chauffeur.ch_code := &Numero_chauffeur ;
new 7:   un_chauffeur.ch_code := 24 ;
Enter value for nom_chauffeur: HAI BON
old 8:   un_chauffeur.ch_nom  := '&Non_chauffeur' ;
new 8:   un_chauffeur.ch_nom  := 'HAI BON' ;

PL/SQL procedure successfully completed.

SQL>
```



30/03/2004

Développement avec PL/SQL

57

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 | | |



30/03/2004

Développement avec PL/SQL

58

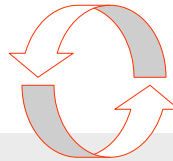
4. Conditions et répétitions

■ Conditions

- Structure **si**: IF-THEN-ELSIF
- Structure **cas**: CASE

■ Répétitions (Boucles)

- Une boucle simple LOOP
- Une boucle FOR
- Une boucle WHILE



30/03/2004

Développement avec PL/SQL

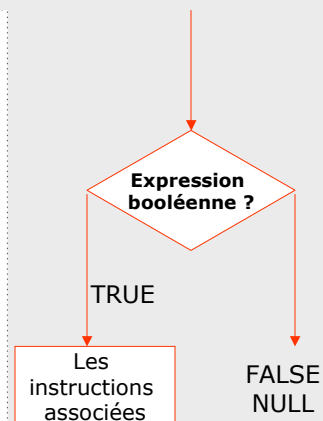
59

4.1 Condition IF

■ Conditions: IF-THEN-END IF

Les expressions booléennes sont la base du contrôle conditionnel qui permettent de contrôler l'exécution d'une séquence d'instructions:

- Si la condition est **TRUE**, les instructions sont exécutées
- Si elle est **FALSE** ou **NULL**, les instructions ne sont pas exécutées



30/03/2004

Développement avec PL/SQL

60

4.1 Condition IF

■ Syntaxe: IF-THEN-ELSIF

IF <condition> THEN
<séquence d'instructions>

[ELSIF <condition> THEN
<séquence d'instructions>]

[ELSE
<séquence d'instructions>]
END IF ;

La <condition> doit aboutir à un résultat booléen: TRUE; FALSE ou NULL

La <condition> est TRUE, la <séquence d'événements> associée est exécutée

Encore l'autre condition ?

La <condition> est FALSE ou NULL, la <séquence d'instructions> alternative est exécutée



30/03/2004

Développement avec PL/SQL

61

4.1 Condition IF

Tables de logique

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL



30/03/2004

Développement avec PL/SQL

62

4.1 Condition IF

- **Exemple 4.1:** Affichez **BONJOUR** s'il est avant 18h. Si non, affichez **BONSOIR**

Conseil: Utiliser **SYSDATE** avec le format 'HH24' : heure de jour (0-23)

```
DECLARE
    heure NUMBER(2);
BEGIN
    SELECT to_char(sysdate,'HH24') INTO heure FROM dual;
    IF heure >= '18' THEN
        DBMS_OUTPUT.PUT_LINE('Bonsoir');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Bonjour');
    END IF;
END;
/
```



30/03/2004

Développement avec PL/SQL

63

4.1 Condition IF

- **Exemple 4.2a:** Affichez le jour de semaine d'aujourd'hui, par exemple 'Aujourd'hui est lundi'

Conseil: Utiliser **SYSDATE** avec le format 'D' : jour de semaine (1-7)

Si à l'intérieur de la chaîne se trouve une apostrophe, celle-ci doit être double

```
DECLARE
    jour_de_semaine NUMBER(2);
    jour_en_caractere CHAR(10);
BEGIN
    SELECT to_char(sysdate,'D') INTO
    jour_de_semaine FROM dual;
    IF jour_de_semaine = 1 THEN
        jour_en_caractere := 'dimanche';
    ELSIF jour_de_semaine = 2 THEN
        jour_en_caractere := 'lundi';
    ELSIF jour_de_semaine = 3 THEN
        jour_en_caractere := 'mardi';
    ELSIF jour_de_semaine = 4 THEN
        jour_en_caractere := 'mercredi';
    ELSIF jour_de_semaine = 5 THEN
        jour_en_caractere := 'jeudi';
    ELSIF jour_de_semaine = 6 THEN
        jour_en_caractere := 'vendredi';
    ELSE -- jour_de_semaine = 7
        jour_en_caractere := 'samedi';
    END IF;
    DBMS_OUTPUT.PUT_LINE
    ('Aujourd'hui est ' || jour_en_caractere);
END;
```



30/03/2004

Développement avec PL/SQL

64

4.1 Structure CASE

■ Structure: CASE ... END CASE

La structure CASE permet d'exécuter d'une séquence d'instructions en fonction de différentes conditions

→ Est utile lorsqu'il faut évaluer *une même expression* et proposer plusieurs traitements pour diverses conditions



30/03/2004

Développement avec PL/SQL

65

4.1 Structure CASE (condition)

■ Syntaxe: CASE -END CASE

```
CASE  
WHEN <condition1> THEN  
    <séquence d'instructions 1>  
  
WHEN <condition2> THEN  
    <séquence d'instructions 2>  
  
[ELSE  
    <séquence d'instructions N>]  
END CASE ;
```

La <condition 1> doit aboutir à un résultat booléen: TRUE; FALSE ou NULL

La <condition 1> est TRUE, la <séquence d'instructions 1> associée est exécutée

Encore l'autre condition ?

Les <condition 1,2,...> est FALSE ou NULL, la <séquence d'instructions> alternative est exécutée



30/03/2004

Développement avec PL/SQL

66

4.1 Structure CASE (condition)

- **Exemple 4.2b:** Affichez le jour de semaine d'aujourd'hui, par exemple 'Aujourd'hui est lundi'

```
DECLARE
  jour_de_semaine NUMBER(2);
  jour_en_caractere CHAR(10);
BEGIN
  SELECT to_char(sysdate,'D') INTO
  jour_de_semaine FROM dual;
  CASE
    WHEN jour_de_semaine = 1 THEN
      jour_en_caractere := 'dimanche';
    WHEN jour_de_semaine = 2 THEN
      jour_en_caractere := 'lundi';
    WHEN jour_de_semaine = 3 THEN
      jour_en_caractere := 'mardi';
    WHEN jour_de_semaine = 4 THEN
      jour_en_caractere := 'mercredi';
    WHEN jour_de_semaine = 5 THEN
      jour_en_caractere := 'jeudi';
    WHEN jour_de_semaine = 6 THEN
      jour_en_caractere := 'vendredi';
    ELSE -- jour_de_semaine = 7
      jour_en_caractere := 'samedi';
  END CASE;
  DBMS_OUTPUT.PUT_LINE
  ('Aujourd'hui est ' || jour_en_caractere);
END;
```



30/03/2004

Développement avec PL/SQL

67

4.1 Structure CASE (expression)

- **Syntaxe:** CASE –END CASE

CASE *sélecteur*

WHEN <expression1> THEN
 <résultat1>

WHEN <expression2> THEN
 <résultat2>

[ELSE
 <résultatN>]

END CASE ;

*Le sélecteur a la valeur
<expression1> ?*

Alors rend le résultat 1

Encore l'autre expression ?

Si non, rend le résultat N



30/03/2004

Développement avec PL/SQL

68

4.1 Structure CASE (expression)

- Exemple 4.2c: Affichez le jour de semaine d'aujourd'hui, par exemple 'Aujourd'hui est lundi'

```
DECLARE
jour_de_semaine NUMBER(2) ;
jour_en_caractere CHAR(10) ;
BEGIN
SELECT to_char(sysdate,'D') INTO
jour_de_semaine FROM dual ;

jour_en_caractere :=
CASE jour_de_semaine

WHEN 1 THEN 'dimanche';
WHEN 2 THEN 'lundi' ;
WHEN 3 THEN 'mardi' ;

WHEN 4 THEN 'mercredi' ;
WHEN 5 THEN 'jeudi' ;
WHEN 6 THEN 'vendredi' ;
ELSE 'samedi' ;

END CASE ;
DBMS_OUTPUT.PUT_LINE
('Aujourd'hui est ' || jour_en_caractere) ;
END ;
```



30/03/2004

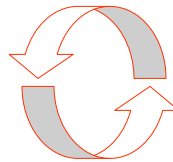
Développement avec PL/SQL

69

4.2 Répétitions

- Une instruction de **boucle** (LOOP) exécute une série d'instructions plusieurs fois consécutives
- Une boucle simple:

```
LOOP
<séquence d'instructions>
EXIT [WHEN <condition>] ;
END LOOP ;
<séquence d'instructions>
```



- L'instruction **EXIT** est utilisée pour sortir d'une boucle et passer le contrôle à l'instruction qui suit **ENDLOOP**



30/03/2004

Développement avec PL/SQL

70

4.2 Répétitions

- Exemple 4.3: Entrer un nombre (N) et calculer la factorielle de ce nombre (N!)

- Explication: Factorielle n (n!) est le produit des nombres consécutifs de 1 à n
 $n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n$

- Le début de la liste:

$$1! = 1$$

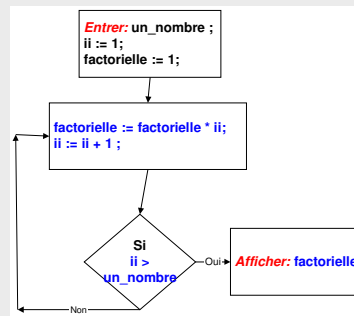
$$2! = 1 \times 2 = 2$$

$$3! = 1 \times 2 \times 3 = 6$$

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

$$N! = (N-1)! \times N$$



30/03/2004

Développement avec PL/SQL

71

4.2 Répétitions

- Exemple 4.3: Entrer un nombre (N) et calculer la factorielle de ce nombre (N!)

```
DECLARE
un_nombre INTEGER;
ii INTEGER ;
factorielle INTEGER;
BEGIN
un_nombre := &ENTRER_UN_NOMBRE ;
ii := 1 ;
factorielle := 1 ;
LOOP
factorielle := factorielle * ii ;
ii := ii + 1 ;
EXIT WHEN ii > un_nombre ;
END LOOP ;
DBMS_OUTPUT.PUT_LINE('Factorielle :'|| TO_CHAR(factorielle));
END ;
```



30/03/2004

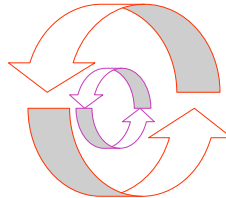
Développement avec PL/SQL

72

4.2 Répétitions

■ La boucle emboîtée (nested loop)

```
LOOP
  <séquence d'instructions 1>
  LOOP
    <séquence d'instructions 2>
    EXIT [WHEN <condition 2>] ;
  END LOOP
  EXIT [WHEN <condition 1>] ;
END LOOP ;
<séquence d'instructions 3>
```



30/03/2004

Développement avec PL/SQL

73

4.2 Répétitions

■ Les boucles avec étiquettes

Avantages:

- Meilleure lisibilité du code;
- Sortie possible de plusieurs boucles imbriquées
 - De la boucle courante
 - De celles qui l'incluent

```
<< étiquette1>>
LOOP
  ....
  << étiquette2>>
  LOOP
    <séquence d'instructions 2>
    EXIT [WHEN <condition 2>] ;
  END LOOP étiquette2
  ....
  EXIT [WHEN <condition 1>] ;
END LOOP étiquette1;
```



30/03/2004

Développement avec PL/SQL

74

4.2 Répétitions

```
BEGIN
  <<boucle_extérieure>>
  LOOP
    nombre := nombre +1;
    EXIT WHEN nombre > 10;

    <<boucle_intérieure>>
    LOOP
      ...
      EXIT boucle_extérieure WHEN total_fait = 'TRUE';
      -- Sortie toutes les deux boucles
      EXIT boucle_intérieure WHEN intérieure_fait = 'TRUE';
      -- Sortie la boucle intérieure seulement

      ....
    END LOOP boucle_intérieure;
  END LOOP boucle_extérieure;
END;
```



30/03/2004

Développement avec PL/SQL

75

4.2 Répétitions

- Une boucle **FOR** numérique exécute la séquence d'instructions un nombre de fois donné

- Une boucle **FOR**:

```
FOR <indice> IN [REVERSE] <entier_min>..<entier_max> LOOP
  <séquence d'instructions>
END LOOP ;
```

- Un **indice (index)** est implicitement déclaré,
- La séquence d'instructions est exécutée autant de fois que l'indice est compris entre **<entier_min>..<entier_max>** ,
- L'indice est incrémenté de 1 (ou -1) à chaque itération.



30/03/2004

Développement avec PL/SQL

76

4.2 Répétitions

■ Exemple 4.4: Une boucle FOR

```
DECLARE
un_nombre INTEGER;
ii INTEGER ;
factorielle INTEGER;

BEGIN
un_nombre := &ENTRER_UN_NOMBRE ;
factorielle := 1 ;

FOR ii IN 1..un_nombre LOOP
    factorielle := factorielle * ii ;
END LOOP ;
DBMS_OUTPUT.PUT_LINE('Factorielle :'|| TO_CHAR(factorielle));
END ;
```



30/03/2004

Développement avec PL/SQL

77

4.2 Répétitions

- Une boucle **WHILE** continue à répéter une séquence d'instructions tant qu'une condition spécifique a la valeur TRUE

- Une boucle **WHILE**:

```
WHILE <condition> LOOP
    <séquence d'instructions>
END LOOP ;
```



30/03/2004

Développement avec PL/SQL

78

4.2 Répétitions

■ Exemple 4.5: Une boucle WHILE

```
DECLARE
un_nombre INTEGER;
ii INTEGER ;
factorielle INTEGER;
BEGIN
un_nombre := &ENTRER_UN_NOMBRE ;
ii := 1 ;
factorielle := 1 ;
WHILE ii <= un_nombre LOOP
factorielle := factorielle * ii ;
ii := ii + 1 ;
END LOOP ;
DBMS_OUTPUT.PUT_LINE('Factorielle :'|| TO_CHAR(factorielle));
END ;
```



30/03/2004

Développement avec PL/SQL

79

4.3 Pratique

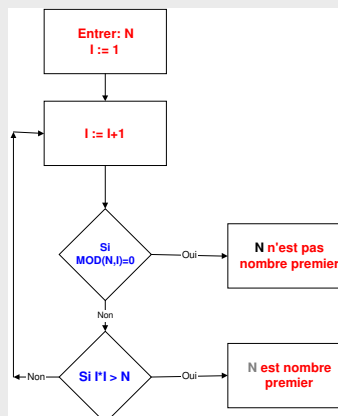
■ Exemple 4.6: Entrer un nombre et informer qu'il est un nombre premier ou pas

- Explication: Un nombre entier plus grand que UN (1) est un nombre premier si ses seuls diviseurs sont 1 et lui-même.

- Exemple:
 $10 = 2 \times 5$ n'est pas premier
 $11 = 1 \times 11$ est premier

- Le début de la liste:
2, 3, 5, 7, 11, 13 ...

■ Test si un nombre (N) est premier



30/03/2004

Développement avec PL/SQL

80

4.3 Pratique

■ Exemple 4.6: Entrer un nombre et informer qu'il est un nombre premier ou pas

```
DECLARE
un_nombre INTEGER; -- nombre à valider
est_nombre_premier BOOLEAN := FALSE;
nombre_temp INTEGER; -- nombre
temporaire
remainder INTEGER; --- integer remainder of
a division
BEGIN
-- Entrer un nombre
un_nombre := &ENTRER_UN_NOMBRE;
nombre_temp := 2;
LOOP
remainder := mod(un_nombre,
nombre_temp);
EXIT WHEN ( remainder = 0) OR
(est_nombre_premier);
nombre_temp := nombre_temp + 1;
IF (nombre_temp * nombre_temp) >
un_nombre THEN
-- Est un nombre premier
est_nombre_premier := TRUE;
END IF;
END LOOP;
IF est_nombre_premier THEN
DBMS_OUTPUT.PUT_LINE('Est un
nombre premier');
ELSE
DBMS_OUTPUT.PUT_LINE('N'est pas
d'un nombre premier');
END IF;
END;
```



30/03/2004

Développement avec PL/SQL

81

4.3 Pratique

■ Exemple 4.6: Entrer un nombre et informer qu'il est un nombre premier ou pas

```
21 ELSE
22   DBMS_OUTPUT.PUT_LINE('N'est pas d'un nombre premier');
23 END IF;
24* END;
SQL> @exemple4_5;
Enter value for entrer_un_nombre: 13
old 8: un_nombre := &ENTRER_UN_NOMBRE;
new 8: un_nombre := 13;
Est un nombre premier

PL/SQL procedure successfully completed.

SQL> @exemple4_5;
Enter value for entrer_un_nombre: 12
old 8: un_nombre := &ENTRER_UN_NOMBRE;
new 8: un_nombre := 12;
N'est pas d'un nombre premier

PL/SQL procedure successfully completed.
```



30/03/2004

Développement avec PL/SQL

82