

Oracle : Langage PL/SQL

1 Introduction à PL/SQL

PL/SQL est un langage de programmation procédural et structuré.

1.1 Langage de programmation

Il contient un ensemble d'instructions permettant de mettre en oeuvre les principes de l'algorithmique.

- Déclaration de variables, de types, de fonctions, de procédures
- Instructions d'affectation, conditionnelles, itératives
- Fonctions numériques et de manipulations de chaînes de caractères, de dates

1.2 Intégration du langage SQL

- Commande SELECT
- Commande INSERT, UPDATE, DELETE
- Commande de gestion de transaction COMMIT, ROLLBACK, SAVEPOINT
- Utilisation de fonctions SQL : TO_CHAR, TO_NUMBER, SUBSTR, ...

1.3 Gestion de curseurs

- Instructions DECLARE, OPEN, FETCH, CLOSE

1.4 Gestion des erreurs

- Gestion standard des erreurs par ORACLE
- Définition et gestion de traitements d'erreurs propres au développeur

1.5 PL/SQL dans le monde ORACLE

- disponible dans les outils de développement ORACLE*FORMS, SQL*plus, ...
- définition de CI, de "triggers", de procédures stockées.

2 Les blocs PL/SQL

Le bloc est l'unité de programmation PL/SQL.

Structure d'un bloc SQL

```
DECLARE
/* Déclaration des variables, des types, des curseurs, fonctions et procédures */
BEGIN
/* Instructions PL/SQL ; tout instruction est terminée par ; */
EXCEPTION
/* Traitement des erreurs */
END ;
```

Les blocs peuvent être imbriqués.

2.1 DECLARE

2.1.1 Déclarations de variables

– Variables simples

Syntaxe

Nom_var [CONSTANT] Type_Donnée [NOT NULL] [:=expr_pl/sql] ;

Exemples :

```
COMPTEUR    Number ;
```

```
TOTAL      Number(8,2) ;
```

```

QUOTA      Number Default 10;

NBR_MIN    CONSTANT Number :=1;

NBR_MAX    Number Default NBR_MIN*QUOTA;

V_NOM      Varchar2(50);

```

```

V_NUM      Number NOT NULL :=1;

```

– Variables %TYPE

Syntaxe

```
Nom_var [CONSTANT] Identifiant%TYPE [NOT NULL] [ :=expr_pl/sql];
```

Déclaration d'une variable faisant référence à l'identifiant. Le type de la variable sera celui de l'identifiant. L'identifiant doit être le nom d'une variable préalablement définie dans le bloc ou le nom d'un attribut d'une table de la base de données.

Exemples :

```
V_COMPT COMPTEUR%TYPE;
```

```
V_NOMP PRIX.NOMP%TYPE;
```

– Variables %ROWTYPE

Syntaxe

```
Nom_var [CONSTANT] Identifiant%ROWTYPE [NOT NULL] [ :=expr_pl/sql];
```

Déclaration d'une variable correspondant à une matrice ayant la même structure (nom et type) que ceux de l'identifiant. L'identifiant doit être le nom d'une variable curseur préalablement définie dans le bloc ou le nom d'une table de la base de données. Un attribut ou champ de la matrice sera accédé par la syntaxe Nom_var.Nom_attribut.

Exemples :

```
V_PRIX PRIX%ROWTYPE;
```

```
V_CURS CURSEUR1%ROWTYPE;
```

– Variables Curseur

Le programmeur ne déclare dans la section DECLARE que les curseurs dits explicites c'est à dire une zone mémoire associée à une requête **select** qui permet de récupérer les n-uplets de la requête.

Syntaxe

```
CURSOR nom_curseur [(nom_paramètre type_paramètre)] IS requête_select [FOR UPDATE OF nom_attribut]
```

Exemples :

```

CURSOR CURSEUR1 IS
    Select nomp,cout
    From Prix
    Where nomf='SAMACO';

```

```

CURSOR Bonclient(V_solde number(8)) IS
    Select nomc,adresc
    From client
    Where solde>V_solde;

```

2.1.2 Déclaration de Types

– Type spécifique RECORD

Syntaxe

```
TYPE Nom_type IS RECORD (nom_champ1 Type_Donnée [NOT NULL], (nom_champ2 Type_Donnée [NOT NULL],...);
```

Type_Donnée tous les types de variables possibles NOT NULL les champs correspondants devront être initialisés

Exemples :

```

TYPE T_Date IS RECORD
    (JOUR INTEGER,
    MOIS INTEGER,
    ANNEE INTEGER);

```

```
V_Date T_Date;
```

```
TYPE T_Client IS RECORD
```

```
(NOM          VARCHAR2(30),
 PRENOM       VARCHAR2(30),
 DATE_NAIS    T_Date);
```

– Type spécifique TABLE

Syntaxe

```
TYPE Nom_type IS TABLE OF type_attribut [NOT NULL], INDEX BY BINARY_INTEGER;
```

```
type_attribut          # Type de données de l'attribut concerné
```

```
NOT NULL              # Ne pas insérer de valeur nulle dans un n-uplet de la table
```

```
INDEX BY BINARY_INTEGER # Création d'un attribut contenant l'indice du
n-uplet de type obligatoire BINARY_INTEGER
```

Exemple :

```
TYPE ADRESSE IS TABLE OF
  CHAR(20)
  INDEX BY BINARY_INTEGER;
```

```
TABLE_ADRESSE ADRESSE;
```

```
INDICE BINARY_INTEGER;
```

2.1.3 Déclaration de Procédures

Les procédures doivent être déclarées en fin de section DECLARE.

Une procédure peut être considérée comme un sous_bloc réalisant un traitement spécifique. Une procédure ne peut être utilisée que dans le bloc qui la déclare. Une procédure est constituée de deux parties : l'en_tête et le bloc de déclaration.

Syntaxe

```
PROCEDURE Nom_procedure [ (Paramètre1, ...) ] IS
```

– déclaration de variables locales

```
BEGIN
```

– instructions [EXCEPTIONS

– gestion des erreurs]

```
END [Nom_procedure];
```

Paramètres

nom_param [IN | OUT | IN OUT] TYPE [:= exp_pl/sql]

IN modifications non prises en compte (par valeur)

OUT valeur en entrée non utilisable

IN OUT modifications prises en compte (par référence)

TYPE tous les types de variable

Exemple :

```
PROCEDURE SUP_Client (N IN Number) IS
```

```
.....
```

```
BEGIN
```

```
/* code de la suppression*/
```

```
END SUP_Client;
```

2.1.4 Fonctions

Les fonctions doivent être déclarées en fin de section DECLARE.

Une fonction peut être considérée comme un sous_bloc réalisant un traitement spécifique retournant une valeur à la fin du traitement. Une fonction ne peut être utilisée que dans le bloc qui la déclare. Une procédure est constituée de deux parties : l'en_tête et le bloc de déclaration.

Syntaxe

```
FUNCTION Nom_fonction [ (Paramètre1, ...) ] RETURN TYPE IS
```

– déclaration de variables locales

```

BEGIN
– instructions [EXCEPTIONS
– gestion des erreurs]
END [Nom_fonction];
Paramètres
nom_param [IN | OUT | IN OUT ] TYPE [ := exp_pl/sql]
IN modifications non prises en compte (par valeur)
OUT valeur en entrée non utilisable
IN OUT modifications prises en compte (par référence)
TYPE tous les types de variable
Exemple :

```

```

FUNCTION TOT_Com (Prix IN Number, QTE IN Number) RETURN REAL IS
.....
BEGIN
/* calcul du total*/
END TOT_Com;

```

2.2 Instructions du corps de bloc

Le corps d'un bloc commence par le mot réservé BEGIN et finit sur le mot réservé END ;

2.2.1 Affectations

L'affectation "classique" :=
L'ordre SELECT

2.2.2 Conditionnelles

```

IF < condition >
THEN < instruction > ;
END IF ;
IF < condition >
THEN < instruction > ;
ELSE < instruction > ;
END IF ;
Condition
< Opérande > < Opérateur > < Opérande >
Opérateurs utilisables
= != <> < <= > >=
IN NOT IN
OR AND NOT

```

2.2.3 Itératives

```

Trois formes
"Label" LOOP
Instructions ;
Test d'arrêt ;
END LOOP Label ;
FOR < variable > IN < Borne_inf > .. < Borne_sup >
LOOP
Instructions ;
END LOOP ;
WHILE < Condition >
LOOP
Instructions ;
END LOOP ;

```

2.2.4 Ordres SQL

On distingue deux groupes d'ordres SQL, celui des ordres "simples" associés à des curseurs implicites, celui des ordres de sélection plus complexes nécessitant des curseurs explicites.

- Les curseurs implicites

Ils sont associés aux ordres SELECT, INSERT, DELETE et UPDATE. Ils sont déclarés automatiquement par ORACLE lors de l'exécution de la requête. (Attention un seul enregistrement doit être résultat pour une requête SELECT)

Variables ORACLE positionnées

`%FOUND %NOTFOUND %ROWCOUNT %ISOPEN`

Utilisation dans les instructions conditionnelles ou d'affectation précédées par SQL

exemple `SQL%FOUND`

Gestion des erreurs `NO_DATA_FOUND` ou `TOO_MANY_ROWS`

- Les curseurs explicites

Les curseurs explicites sont utilisés en quatre phases : - Déclaration - Ouverture (OPEN) - Lecture d'un enregistrement (FETCH INTO) - Fermeture (CLOSE)

Variables ORACLE positionnées

`%FOUND %NOTFOUND %ROWCOUNT %ISOPEN`

Utilisation dans les instructions conditionnelles ou d'affectation

précédées par le nom du curseur associé

exemple `CURSEUR1%NOTFOUND`

- Les ordres SQL relatifs aux transactions

`COMMIT ROLLBACK SAVEPOINT`

3 Notion de Package

Un Package est un ensemble nommé d'objets associés (TYPES, VARIABLES, FONCTIONS, PROCEDURES) stocké dans la base.

Un Package comporte deux parties

- une partie SPECIFICATION visible depuis les applications externes

```
CREATE [OR REPLACE] PACKAGE [Schéma.]Nom_package IS Spécification_pl/sql
```

- une partie CORPS invisible depuis les applications externes

```
CREATE [OR REPLACE] PACKAGE BODY [Schéma.]Nom_package IS pl/sql_package_body
```

L'exécution d'une procédure d'un package s'effectue par la commande

```
EXECUTE nom_package.nom_procedure(liste_paramètres effectifs)
```