II- PL/SQL : Extension au standard SQL

1. Introduction

Le langage SQL n'est pas un langage de programmation au sens courant du terme. Il ne permet pas de définir des **fonctions** ou des **variables**, d'effectuer des **itérations** ou des **instructions conditionnelles. Le langage SQL** ne suffit pas pour le développement des applications.
Tous les SGBDR proposent des interfaces permettant d'associer le langage SQL à des langages plus classiques comme le C ou Java. Ces interfaces de programmation permettent d'utiliser SQL comme outil pour récupérer des données dans des programmes réalisant des tâches très diverses : interfaces graphiques, traitements « batch », production de rapports ou de sites web, etc.
Pourtant, le recours à un langage de programmation « externe » pour certaines fonctionnalités, s'avère inadapté ou insatisfaisant. D'où, la nécessité de proposer, au sein même du système, des primitives de programmation pour pallier ce manque relatif d'expressivité.
Dans cette optique, l'utilisation de PL/SQL est indiquée :

- Pour effectuer des traitements de données qui impliquent des transactions complexes (plusieurs requêtes et manipulations de données liées).
- Pour effectuer un contrôle d'intégrité des données (Triggers par exemple).
- Pour stocker dans la BD les opérations fréquentes. (Packages et procédures stockées.)
- Pour minimiser le temps d'interaction entre la BD et la partie interface.
- Pour minimiser l'impact des MAJ sur une application avec un grand nombre de postes clients.

2. PL/SQL : Structure

Le PL de PL/SQL signifie « Procedural Language » est une extension procédurale du SQL permettant d'effectuer des traitements complexes sur une base de données. Les possibilités offertes sont les mêmes que les langages impératifs. Ainsi, tout code écrit dans un langage procédural est formé de Blocs. Chaque bloc comprend une section de déclaration de variables, et un ensemble d'instructions dans lequel les variables déclarées sont visibles.

```
DECLARE
        /* Déclaration des variables */
BEGIN
        /* Instructions à exécuter */
EXCEPTION
        /* Traitement des erreurs */
END ;
```

Pour résumer, un programme ou une procédure PL/SQL est constitué d'un ou plusieurs blocs avec les trois sections suivantes :

- Déclaration des structures et des variables utilisées dans le bloc (Facultative)
- Corps qui contient les instructions (Obligatoire)
- Traitement des erreurs : pour gérer les erreurs. (Facultative)
- Un script écrit en PL/SQL se termine obligatoirement par un /, sinon SQL+ ne l'interprète pas.

a. Déclaration des variables

En Général : La déclaration d'une variable sous Oracle se fait selon la syntaxe suivante :

nom_variable [CONSTANT] Type [[NOT NULL] := Expression]

- **CONSTANT** : le bloc ne changera pas la valeur
- **Expression** : pour initialiser la variable
- **NOT NULL** impose une initialisation
- **Type** : le type de la variable

Pour afficher le contenu d'une variable, on utilise les procédures DBMS_OUTPUT.PUT() et DBMS_OUTPUT.PUT_LINE() prennent en argument une valeur pour l'afficher.
Par défaut, les fonctions d'affichage sont désactivées : Il convient de les activer avec la commande SQL+ : SET SERVEROUTPUT ON.

b. Types des variables :

Le type d'une variable peut être simple, dérivé, sous type défini ou complexe

Types simples :

- ✓ Numériques : NUMBER[(e,d)], BINARY INTEGER, BINARY FLOAT
- ✓ Alphanumériques : CHAR, VARCHAR2, LONG, ...
- ✓ Date : DATE, TIMESTAMP, ...
- ✓ Booléen : BOOLEAN
- ✓ Une adresse d'une ligne dans une table : **ROWID**

Types dérivés :

- ✓ Référence à une colonne ou à un type existant
  - nomVariable nomTable.nomColonne%TYPE
  - nomVariable nomVariableRef%TYPE
- ✓ Référence à une ligne d'une table ou d'un curseur
  - nomVariable nomTable%ROWTYPE
  - nomVariable nomCurseur%ROWTYPE

Exemples de types dérivés :

- Quel est le format du champs NomIndividu dans une table Individu ?
  nomActeur Individu.NomIndividu%TYPE;
- Quels sont les formats des attributs de la table Film ?
  film Film%ROWTYPE;

Types définis :

- ✓ Sous-type d'un type existant, avec **SUBTYPE**. Exemples :
  - SUBTYPE chaineCourte IS VARCHAR2(20);
  - SUBTYPE nomPersonne IS Individu.NomIndividu%TYPE;
  - SUBTYPE lm IS Film%ROWTYPE;
- ✓ Type complexe : **TABLE, RECORD, VARRAY**, avec TYPE. (Détails dans la section: II.3)
  - Exemples : TYPE numberTab IS VARRAY (10) OF NUMBER;
  - TYPE tableFilms IS TABLE OF Film;
  - TYPE point IS RECORD (abscisse NUMBER, ordonnee NUMBER);

c. Affectation des variables

L'affectation d'une variable peut être simple (**nomVariable := valeur**) ou bien en utilisant la syntaxe **SELECT ... INTO** quand le SELECT ne retourne qu'une ligne. **Syntaxe:**

SELECT listeAttributs INTO listeVariables
FROM...

OU BIEN

SELECT listeAttributs INTO nomStructure
FROM...

Exemple: ***SELECT * INTO film FROM Film WHERE NumFilm=1;***

3. PL/SQL : Structures de contrôle

a. Traitements conditionnels

Pour un traitement conditionnel, on fait appel aux instructions (IF ou CASE). Elles fonctionnent de la même façon que dans les autres langages impératifs. Les conditions sont les mêmes qu'en SQL

```
IF      /* condition1 */   THEN
        /* instruction1 */
ELSE    /* instruction2 */
END IF;
```

Avec ELSIF :

```
**************************
IF      /* condition1 */      THEN
        /* instruction1 */ ;
ELSIF   /* condition1 */      THEN
        /* instruction */;
ELSE    /* instruction3 */;
END IF;
```

Exemple:

```
**************************
IF X > 0 THEN
        Y: = 'POSITIF';
ELSIF X < 0 THEN
   Y: = 'NEGATIF';
ELSE
   Y: = 'NUL';
END IF;
```

Le switch du langage C s'implémente en PL/SQL de la façon suivante :

```
CASE    /* variable */
WHEN    /* valeur 1 */          THEN
        /* instructions 1 */
WHEN    /* valeur 2 */          THEN
        /* instructions 2 */
...
WHEN    /* valeur n */          THEN
        /* instructions n */
ELSE    /* instructions par défaut */
END CASE;
```

b. Traitements répétitifs

b.1. LOOP ...EXIT WHEN

En PL/SQL, l'instruction **LOOP... END LOOP** permet d'implémenter une boucle. L'instruction **EXIT WHEN** permet de quitter une boucle. **SYNTAXE** :

```
LOOP          /* instructions */
EXIT WHEN     /* condition */;
              /* instructions */
END LOOP;
```

Ou bien :

```
**************************
LOOP          /* instructions */
IF            /* condition */ ; THEN EXIT ;
              /* instructions */
END LOOP;
```

b.2. La boucle FOR

La syntaxe de la boucle FOR en PL/SQL est la suivante :

```
FOR     /* variable */   IN [REVERSE] /* Inf */ . . /* Sup */
LOOP
                /* instructions */
END LOOP;
```

Avec:

- **Variable :** variable locale à la boucle, non déclarée
- **Inf et Sup :** variables déclarées et initialisées ou constantes
- **Pas de boucle est fixé à 1, éventuellement en décrément si : REVERSE**

Exemple : Dans cet exemple, la variable i commence par 1 et s'incrémente à chaque itération de 1 jusqu'à 5 puis on sort de la boucle. Le résultat de X sera 5.

```
DECLARE
        X: = 0;
FOR i IN 1 .. 5 LOOP
        X: = i;
END LOOP;
DBMS_OUTPUT.PUT_LINE (X| | ' La valeur est :' | | X);
END;
/
```

b.3. La boucle WHILE

La syntaxe de la boucle WHILE en PL/SQL est la suivante :

```
While       /* condition */      LOOP
            /* instructions */
END LOOP ;
```

Les instructions sont répétées tant que la condition est vraie.

Exemple : Ecrire un programme plaçant la valeur 10 dans une variable a, puis affichant la factorielle de : a.

```
DECLARE
        a NUMBER;
        res NUMBER;
        counter NUMBER;
BEGIN
        a: = 10;
        res: = 1;
        counter: = a;
WHILE   counter > 0  LOOP
        res : = res * counter ;
        counter: = counter – 1;
END LOOP;
DBMS_OUTPUT.PUT_LINE (a | | '! =' | | res);
END;
/
```

4. Tableaux et Structures

La gestion des tableaux et structures demeure une tâche importante dans le langage PL/SQL.

a. Les tableaux

a.1. Création d'un type tableau

Les types tableau doivent être déclaré explicitement selon la syntaxe suivante :

TYPE /* type */ IS VARRAY (/* taille */) OF /* typeElements */;

- **Type** est le nom du type tableau créé par cette instruction
- **Taille** est le nombre maximal d'éléments qu'il est possible de placer dans le tableau.
- **TypeElements** est le type des éléments qui vont être stockés dans le tableau

Exemple: ***TYPE numberTab IS VARRAY (10) OF NUMBER;***

La déclaration d'un tableau des entiers numberTab est faite comme suit :

```
DECLARE TYPE numberTab IS VARRAY (10) OF NUMBER;
        t numberTab;
BEGIN
      /*instructions*/
END ;
/
```

a.2. Allocation d'un tableau

La création d'un type tableau met à disposition un constructeur du même nom. Si un type tableau numberTab est créé, la fonction numberTab() retourne une tableau vide.

```
DECLARE TYPE numberTab IS VARRAY (10) OF NUMBER;
        t numberTab;
BEGIN
      t : = numberTab ( ) ;
      /* utilisation du tableau */
END ;
/
```

a.3. Dimensionnement d'un tableau

Le tableau retourné par le constructeur est vide. Il convient ensuite de réserver de l'espace pour stocker les éléments en utilisant la méthode **EXTEND ().**
Dans l'exemple suivant, t.EXTEND(4) permet d'utiliser les éléments t(1), t(2), t(3) et t(4). Il n'est pas possible d'étendre un tableau à une taille supérieure à celle spécifiée lors de la création du type tableau associé (ici : 10).

```
DECLARE TYPE numberTab IS VARRAY (10) OF NUMBER;
        t numberTab;
BEGIN
      t: = numberTab ();
      t. EXTEND (4);
      /* utilisation du tableau */
END;
/
```

a.4. Utilisation d'un tableau

On accède, en lecture et en écriture, au i-ème élément d'une variable tabulaire nommé T avec l'instruction T(i). Les éléments sont indicés à partir de 1.
Exemple : Une permutation circulaire vers la droite des éléments d'un tableau t.

```
DECLARE
        TYPE numberTab IS VARRAY (10) OF NUMBER;
        t numberTab;
        i number;
        k number;
BEGIN
t: = numberTab ();
t.EXTEND (10) ;
        FOR i IN 1..10 LOOP
        t (i) := i ;
        END LOOP;
k: = t (10);
        FOR i in REVERSE 2..10 LOOP
        t (i) := t ( i – 1 ) ;
        END LOOP;
t (1):= k ;
        FOR i IN 1 . . 10 LOOP
        DBMS_OUTPUT.PUT_LINE (t(i));
        END LOOP;
END;
/
```

b. Les structures

Une structure est un type regroupant plusieurs types. Une variable de type structuré contient plusieurs variables, ces variables s'appellent aussi des champs.

b.1. Création d'un type structuré

```
TYPE   /* NomType */    IS RECORD
(
        /* Liste des champs */
) ;
```

– **NomType** : est le nom du type structuré crée par cette instruction
– **Listedeschamps**: Même syntaxe que la liste des dans un CREATE TABLE

Exemple:

```
TYPE point IS RECORD
(
abscisse NUMBER,
ordonnee NUMBER
) ;
```

b.2. Déclaration d'un type structuré

Maintenant, il possible de créer des variables de type point.

- **p point ;** permet de déclarer une variable p de type point

b.3. Tables et structures

On rappelle que pour placer dans une variable le résultat d'une requête, on utilise le mot-clé INTO. Les instructions suivantes permettent d'affecter aux variables v_1, ..., v_n les valeurs retournées par la requête. La requête doit retourner une et une seule ligne, sinon, une erreur se produit à l'exécution.

```
SELECT    champ_1, . . . , champ_n
INTO      v_1, . . . , v_n
FROM ...
```

Exemple :

```
DECLARE
      num NUMBER;
      nom VARCHAR2(30) := 'Mon_produit' ;
BEGIN
      SELECT numprod INTO num
        FROM PRODUIT
        WHERE nomprod = nom ;
        DBMS_OUTPUT.PUT_LINE ('L' 'article' || nom || 'a pour numero '|| num ) ;
END;
/
```

Toutefois, on peut utiliser **nomTable.nomColonne%type** pour se référer directement au type d'une colonne. L'exemple précédent devient :

```
DECLARE
        num PRODUIT . numprod%type ;
        nom PRODUIT . nomprod%type := 'Mon_produit';
BEGIN
      SELECT numprod INTO num
        FROM PRODUIT
        WHERE nomprod = nom;
        DBMS_OUTPUT.PUT_LINE ('L' 'article' || nom || 'a pour numero '|| num ) ;
END;
/
```

Il est possible de déclarer **une structure** pour représenter une ligne d'une table, le type porte alors le nom suivant : **nomTable%rowtype.**

Exemple :

```
DECLARE
   nom PRODUIT. nomprod%type := 'Mon_produit';
   ligne PRODUIT%rowtype ;
BEGIN
   SELECT * INTO ligne
   FROM PRODUIT
WHERE nomprod = nom;
DBMS_OUTPUT.PUT_LINE ('L''article' || ligne.nomprod || 'a pour num' ||
ligne.numprod ) ;
END;
/
```