PL/SQL

I. Preambule

SGBD Relationnel

PL/SQL : **Procedural Language** (Oracle)

Remarque : Outils pour faire du P/SQL

LiveSQL ; OEE (oracle express edition (v10) c'est un exe) ; téléchargement sur oracle ; SQLDevelopper.

Définition : Le PL/SQL est une extension du SQL (SELECT, INSERT, DELETE, UPDATE) auquel on vient adosser du code

Code : structure répetitive (boucles), structure alernative (if….), recursivité…

SQL niveau 1 : on évolue dans un environnement CLIENT / SERVEUR => tous les échanges impliquent une réponse en retour donc des échanges entre le C et le S.

SQL niveau 2 : on évolue dans un environnement CLIENT / SERVEUR avec un BLOC qui ne donne lieu qu'à UN SEUL ECHANGE

Bloc : c'est une suite d'instructions contenues dans un script dans lequel on retrouve du SQL et du code.

SQL 1 niveau 1

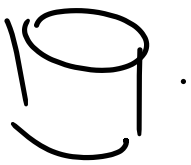
CLIENT SERVEUR

Select

Update

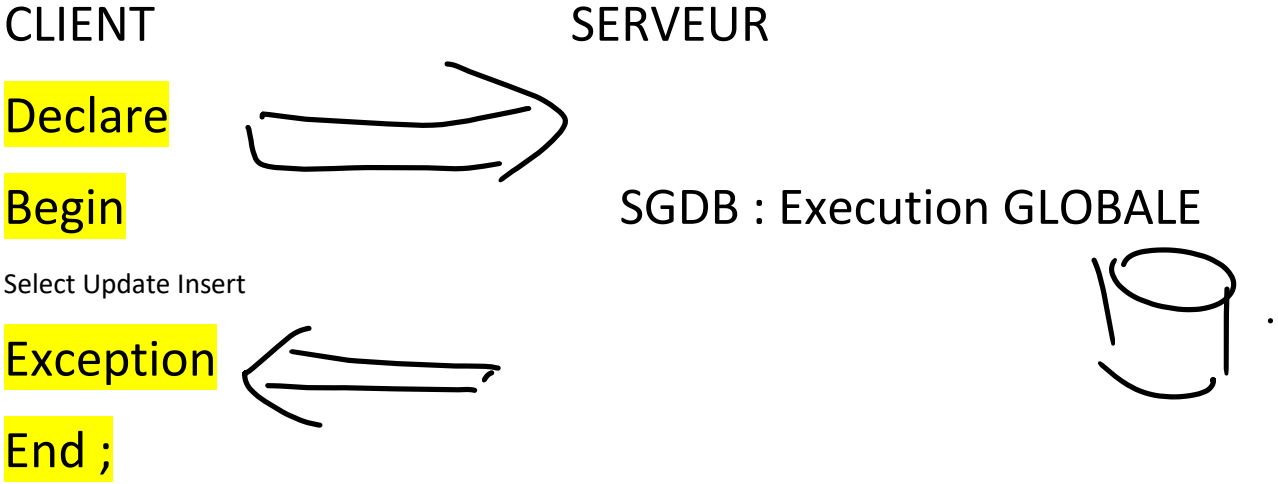
Insert

SGBD : execution requete par requete



.

SQL 2 niveau 2 **: un seul échange entre le client et le serveur (avec les résultats intermédiaires qui sont calculés coté serveur)**



Structure du bloc (script, programme) :

Le bloc est organisé en sections : DECLARE, BEGIN, EXCEPTION, END

DECLARE : on déclare les variables, les types, les curseurs……

BEGIN : section obligatoire qui contient toutes les directives SQL … On peut faire appel à d'autres blocs depuis ce bloc

EXCEPTION permet de traiter les erreurs et de les faire remonter

END ;

Remarques :

-pas de casse comme en SQL (A….Z, a….z, 0…..9) commentaires avec – ou /**/

II. Variables

On peut en PL/SQL manipuler des variables => on peut transmettre des valeurs et réaliser l'affichage en sortie de résultats.

Il existe différent types de variables :

-variable de base :

Identificationdunomdelavariable typededonnée ;

/*déclaration de la variable nbre*/

nbre NUMBER (3) ;

/*déclaration de la variable nbre et initialisation à 999*/

nbre NUMBER (3) := 999 ;

/*déclaration de la variable x*/

x NUMBER (8) ;

/*déclaration de la variable x et initialisation à 125*/

x NUMBER (3) := 125 ;

remarque : on peut mettre un v devant la variable pour expliquer que c'est une variable par convention

v_datenaissance DATE ;

v_trouv BOOLEAN ;

-variable %TYPE :

La directive %TYPE permet de déclarer une variable selon la définition **d'une colonne** d'une table (ou d'une VUE)

Cf exemple : on se trouve dans la partie DECLARE. On a une BD avec une table PILOTE qui a pour propriété BREVET

V_brevet Pilote.brevet%TYPE;

-variable %ROWTYPE :

On travaille sur un enregistrement (record), la directive **ROWTYPE permet de définir un ensemble de colonnes ou toutes les colonnes d'une table.**

Cf exemple : on se trouve dans la partie DECLARE. On a une BD avec une table PILOTE qui a différentes colonnes

Exple1 : Rty_pilote Pilote.%ROWTYPE; cette variable est liée à tout ou partie des colonnes de la table pilote

Exple2 :

Pour un select :

```
SELECT  *
INTO Rty_pilote
FROM Pilote
WHERE brevet = 'PL_1';
```

Pour une affectation

```
Rty_pilote.brevet := XXXXXXX;
Rty_pilote.nom := YYYYYY;
```

Pour un insert

```
INSERT INTO
VALUES rty_pilote
```

-variable de substitution :

Avec une variable de substitution on va passer en paramètre d'entrée de bloc **avant** le BEGIN.

On va accéder aux valeurs d'une variable dans le code (on va faire préfixer cette variable du symbole &)

Exemple : on utilise deux variables avant le DECLARE qui n'ont pas à être déclarées.

Interaction avec l'utilisateur.

Remarque : utilisation d'un package d'affichage DBMS_OUTPUT.PUT_LINE

-variable de session :

Une variable de session est une variable globale que l'on retrouve avant le DECLARE.

Il faut faire préfixer le nom de la variable de session du symbole :

On utilise cette variable de session avec les mêmes options que les variables de bases dans le BEGIN

-variable tableaux :

Il s'agit du type TABLE => on va manipuler des tableaux DYNAMIQUES (taille sans dimension initiale).

Le tableau est composé d'une clé primaire et d'une colonne (qui a un type de base, %TYPE, %ROWTYPE…….) qui stocke chaque élément.

Déclaration d'un tableau :

*TYPE nomdutableau IS TABLE OF variable de base, %TYPE, %ROWTYPE /*type de variable du contenues dans le tableau*/*

INDEX BY BINARY INTEGER

Exemple : on va définir le tableau puis on déclare des variables liées à ce tableau dans la partie DECLARE

Tab_brevets brevet_tytab /*on déclare un tableau tab_brevets lié à brevet_tytab*/

Remarque : il existe des fonctions préfédinies qui effectuent des actions COUNT, EXIST …..

Remarque : on peut utiliser des indices pour affecter des valeurs à ce tableau. Par exemple (O, -1, -2, 7800) => l'accès à ces éléments se fait via un pointeur.

-Affection de variables

Il existe plusieurs possibilités pour affecter une valeur à une variable :

- Variable := affectation ou expression
- DEFAULT exple : v_trouv BOOLEAN DEFAULT FALSE ;
- Utilisation de la directive INTO dans le cadre d'une requête :

```
SELECT affichage
INTO variable
FROM
WHERE
;
```

Exple :

```
DECLARE /*declaration de variable*/
v_brevet VARCHAR(6);

BEGIN
SELECT brevet
INTO v_brevet  /*affectation d'une chaine de caractère par une requete*/
FROM pilote
WHERE nom = 'toto';
```

III. Interaction avec la Base de Données

Il s'agit des mécanismes pour intefacer un script PL/SQL avec la BD

- **Extraire des données**

  On utilise :

  SELECT

  INTO

  FROM

  WHERE

- **Manipuler des données**

  Il s'agit : INSERT UPDATE DELETE

- **Paquetage DBMS_OUTPUT**

  Ce paquetage permet d'afficher des réusultats.

  DBMS_OUTPUT.PUT_LINE

  Remarque : c'est utile aussi pour afficher des résultats intermédiaires

  Exple : affichage de resultats

  DBMS_OUPUT.PUT_LINE ('CHAINE DE CARACTERE' ||variable|| 'AUTRE CHAINE DE CARACTERE');

IV. Séquences

Définition : On parle d'objet pour la séquence (objet au sens oracle) (objet au sens système et objet utilisateur).
On effectue une action à un moment donné : l'objectif est de générer une suite lié à des entiers (lien avec PK)
La sequence peut être liée à toutes les actions en SQL.

Code :
Créer => CREATE :

```
CREATE SEQUENCE nomsequence
                    INCREMENT BY  (incrementation de la séquence=> entier)
                    START WITH (entier)
                    MAXVALUE
                    MINVALUE
```

Modifier :

```
ALTER SEQUENCE nomsequence
```

Supprimer :

```
DROP SEQUENCE nomsequence
```

Fonction :
On associe la sequence à une fonction : **sequence.fonction**
CURRVAL : retour de la valeur courante de la sequence
NEXTVAL : incrementation de la sequence

Exemple :

```
CREATE SEQUENCE nomsequence  /*nom de la sequence*/
                    INCREMENT BY 1 /*indentation non necessaire*/
```

```
INSERT INTO client /*creation d'une ligne supplémentaire via l'insert*/
                    VALUES (nomsequence.NEXTVAL,'durant','P12',SYSDATE)

SELECT nomsequence.CURRVAL   /*cela nous retourne un resultat : 1*/
FROM client
```

V. SouS-Programmes

Il existe deux types de sous-programmes :

Procédure : pour effectuer une action (seule la procédure peut avoir des paramètres en sortie). (cf document ci-dessous)

Fonction : renvoyer un résultat. (cf document ci-dessous)

Remarque : les sous-programmes sont compilés et on retrouve leur code (ces programmes peuvent être utilisés, partagés dans un cadre multi-utilisateurs) dans le dictionnaire de données.

le noyau recompile (lors de l'appel de la fonction ou de la procédure) le programme si un objet cité dans le code a été modifié et le charge en mémoire

En terme d'utilisation : il existe des avantages à utiliser des sous-programmes en terme de sécurité (gestion des objets avec les droits d'accès sur les programmes stockés), d'intégrité (les traitements dependants sont executés dans la même transaction), de performance (simplicité en terme d'utilisation et de maintenance) et de productivité (limitation du nombre d'appels à la base)

Transaction : c'est une suite de code qui permet de faire passer la base de données d'un état cohérent à un autre état cohérent lui aussi. (atomicité de l'instruction contenue dans la transaction)

Appel : la personne qui a les droits pour gérer les sous-programmes peut les executer (priviliège pour lancer : EXECUTE). Le mode de fonctionnement est identique que ce soit une procédure ou une fonction.

Supprimer : DROP (DROP FUNCTION nomssprog ; DROP PROCEDURE nomssprog)

Exemple :

Creation de la Procedure :

```
CREATE [OR REPLACE] PROCEDURE [schéma.]nomProcédure
        [(paramètre [ IN | OUT | IN OUT ] [NOCOPY] typeSQL
              [{:= | DEFAULT} expression]
        [,paramètre [ IN | OUT | IN OUT ] [NOCOPY] typeSQL
              [{:= | DEFAULT} expression]... ) ] ]
{ IS | AS } corpsduSousProgrammePL/SQL  ;
```

Creation de la Fonction :

tions cataloguées

```
CREATE [OR REPLACE ] FUNCTION [schéma.]nomFonction
          [(paramètre [ IN | OUT | IN OUT ] [NOCOPY] typeSQL
                 [{:= | DEFAULT} expression]
          [,paramètre [ IN | OUT | IN OUT ] [NOCOPY] typeSQL
                 [{:= | DEFAULT} expression]... ) ] ]
 RETURN typeSQL
   {IS | AS}  corpsduSousProgrammePL/SQL  ;
```

- il doit se trouver une instruction RETURN dans le code.

Gestion des paramètres IN, OUT, IN OUT

- `IN` désigne un paramètre d'entrée, `OUT` un paramètre de sortie et `IN OUT` un paramètre d'entrée et de sortie. Il est possible d'initialiser chaque paramètre par une valeur.
- `NOCOPY` permet de transmettre directement le paramètre. On l'utilise pour améliorer les performances lors du passage de volumineux paramètres de sortie comme les `RECORD`, les tables `INDEX-BY` (les paramètres `IN` sont toujours passés en `NOCOPY`).
- *corpsduSousProgrammePL/SQL* contient la déclaration et les instructions de la procédure, toutes deux écrites en PL/SQL.

Appel de Procedure et de Fonction :

Procédure	Fonction

SET SERVEROUT ON
DECLARE
v_comp VARCHAR2(4) := 'AF';
v_nom VARCHAR2(16);
v_heuresVol NUMBER(7,2);
BEGIN
PlusExpérimenté(v_comp,
v_nom, v_heuresVol);
DBMS_OUTPUT.PUT_LINE
('Nom, heures de vol ' || v_nom ||
' : ' || v_heuresVol);
END;
/
	SET SERVEROUT ON
	DECLARE
	v_comp VARCHAR2(4) := 'AF';
	v_heuresVol NUMBER(7,2) := 300;
	v_résultat NUMBER;
	BEGIN
	v_résultat :=
	EffectifsHeure(v_comp,v_heuresVol);
	DBMS_OUTPUT.PUT_LINE('Pour AF et
	300h résultat : ' || v_résultat);
	END;
	/

Nom, heures de vol Gilles Laborde : 2450 Procédure PL/SQL terminée avec succès.	Pour AF et 300h résultat : 2 Procédure PL/SQL terminée avec succès.