PL/SQL

I. Preambule

SGBD Relationnel

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

Remarque : Outils pour faire du PL/SQL

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

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

Code : structure répétitive (boucles), structure alternative (if….), récursivité…

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

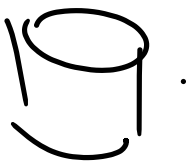
CLIENT SERVEUR

Select

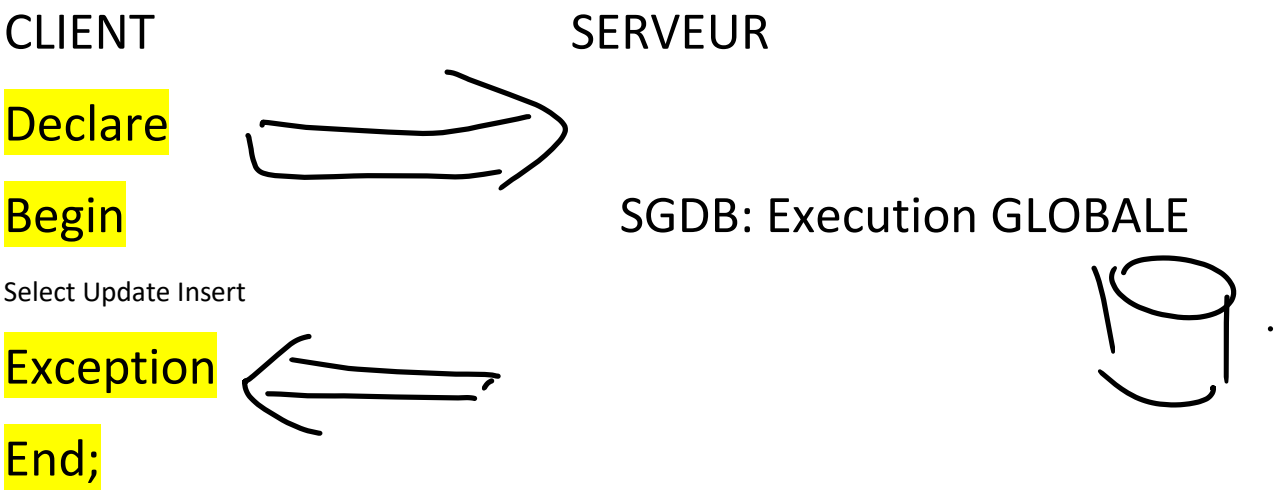
Update

Insert

SGBD : exécution requête par requête



SQL 2 niveau 2 : **un seul échange entre le client et le serveur (avec les résultats intermédiaires qui sont calculés côté Serveur et seul le résultat final est renvoyé au Client)**



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, le code PL/SQL … On peut faire appel à d'autres blocs depuis ce bloc

EXCEPTION permet de traiter les erreurs retournées par le SGBD et de les faire remonter

END ;

Remarques :

- pas de casse comme en SQL (A….Z, a….z, 0…..9), les symboles, les commentaires avec – ou /**/. Un programme en PL/SQL interprète tous les caractères.

II. Variables

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

Il existe différents types de variables :

-Variable de base :

Identificationdunomdelavariable typededonnée ;

/*déclaration de la variable test*/

test NUMBER (3) ;

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

test 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 existante)

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; /*V_brevet prend le type de la colonne brevet de la table Pilote*/ /*on utilise un V majuscule ou V minuscule => PAS de Casse*/

-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.**

Ce type de variable est très utilisé notamment pour la gestion et l'administration de la base de données

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 Pilote /*on travaille dans la table pilote*/
VALUES rty_pilote
```

-Variable de substitution :

Avec une variable de substitution on va passer en paramètre en entrée de bloc **avant le BEGIN** => via l'utilisation du ACCEPT.

ACCEPT S_brevet PROMPT 'blablalblalbla' ; /*S en rapport avec la variable de substitution par convention. Nous n'avons pas à les DECLARER */

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

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

Interaction avec l'utilisateur.

Remarque : utilisation d'un package d'affichage via 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 : dans le code

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 prédéfinies 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 depuis brevet*/
FROM pilote
WHERE nom = 'toto';
```

III. Interaction avec la Base de Données

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

- **Extraire des données**

  On utilise le SELECT :

  SELECT

  INTO

  FROM

  WHERE

- **Manipuler des données**

  Il s'agit : INSERT UPDATE DELETE

- **Paquetage DBMS_OUTPUT**

  Ce paquetage permet d'afficher des résultats.

  DBMS_OUTPUT.PUT_LINE

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

  Exple : affichage de résultats

  DBMS_OUTPUT.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 liée à des entiers (lien avec PK)
La séquence peut être liée à toutes les actions en SQL.

Code :
Créer => CREATE :

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

Modifier :

```
ALTER SEQUENCE nomsequence
```

Supprimer :

```
DROP SEQUENCE nomsequence
```

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

Exemple :

```
CREATE SEQUENCE nomsequence  /*nom de la séquence*/
                INCREMENT BY 1 /*indentation non nécessaire*/



INSERT INTO client /*création d'une ligne supplémentaire via l'insert*/

                VALUES (nomsequence.NEXTVAL,'durant','P12',SYSDATE)


SELECT nomsequence.CURRVAL   /*cela nous retourne un résultat : 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-utilisateur) 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 termes 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 dépendants sont exécutés 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 exécuter (privilège 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 :

Création de la Procédure :

```
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  ;
```

Création 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.

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.

VI. Curseurs

Appel de Procédure et de Fonction :

Lien entre Curseurs et PL/SQL

Objectifs :

Traitement d'un ensemble de résultats via la requête et pas depuis espace mémoire client. Consommation mémoire coté serveur.

Rmque : curseur explicites et implicites (non déclarés).

Structure d'un Programme :

DECLARE

BEGIN

EXCEPTION

END ;

Le curseur se déclare dans la partie DECLARE

```
CURSOR NOM IS
SELECT blablabl ;

-- Déclaration de la variable
v_monCurseur NOM%ROWTYPE; --on peut stocker les informations
```

Utilisation du curseur

OPEN ouvrir le curseur pour structurer la requête. On bloque à ce moment-là. (Réservation en vue de mise à jour)

FETCH parcourir le résultat et à chaque boucle envoie des résultats

CLOSE fermeture

Exemple

```
CREATE OR REPLACE PROCEDURE CurseurMatchs (p_annee INT) AS

 -- Déclaration d'un curseur paramétré
 CURSOR MonCurseur (v_annee INTEGER) IS
  SELECT idmatch, nomrencontre, prenom, nom
  FROM Match, Joueurs
  WHERE idMS = idJoueurs
  AND   annee = v_annee;

 -- Déclaration de la variable associée au curseur
 v_monCurseur MonCurseur%ROWTYPE;
 -- Déclaration de la variable pour la liste des joueurs
 v_mesJoueurs VARCHAR(255);

BEGIN
 -- Ouverture du curseur
 OPEN MonCurseur(p_annee);

 -- On prend le premier n-uplet
 FETCH MonCurseur INTO v_monCurseur;
 -- Boucle sur les n-uplets
 WHILE (MonCurseur%FOUND) LOOP
  -- Recherche des joueurs avec la fonction MesJoueurs
  v_mesJoueurs := MesJoueurs (v_monCurseur.idMatch);

  DBMS_OUTPUT.PUT_LINE('Journée ' || MonCurseur%ROWCOUNT ||
       ' Match:  ' || v_monCurseur. nomrencontre ||
       ', avec ' || v_monCurseur.prenom || ' ' ||
       v_monCurseur.nom || ', et ' || v_mesJoueurss);
  -- Passage au n-uplet suivant
  FETCH MonCurseur INTO v_monCurseur;
 END LOOP;
 -- Fermeture du curseur
 CLOSE MonCurseur;

 EXCEPTION
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE('Problème dans CurseurMatchs, tous les matchs ont été gagnés : ' ||
                  sqlerrm);

END;
/

SQL> set serveroutput on
SQL> execute CurseurMatchs (2019);
Journée 1 Match:  XXX YYY, titi, riri, fifi, loulou
Journée 2 Match:  ZZZ XXX, toto, tata, titi, riri
```

VII. Exceptions

Objectifs :
Eviter l'arrêt d'un programme (division par 0, valeur incorrecte, erreur dans la base….)

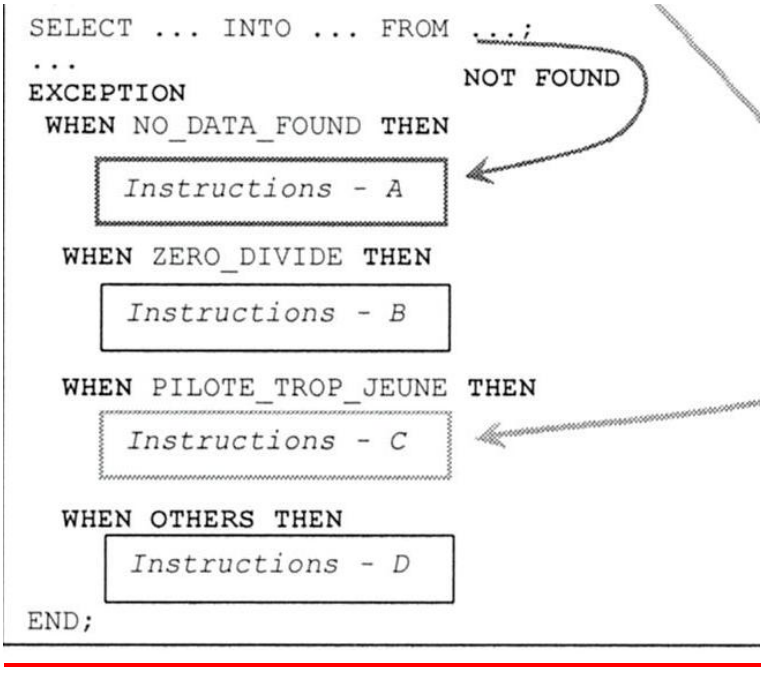
Il est indispensable de prévoir des cas d'erreurs.

En PL/SQL, une exception correspond à une condition d'erreur et on lui associe un identificateur : programme d'erreur exception.

Deux mécanismes peuvent générer une exception :
- Erreur Oracle : exception déclenchée automatiquement (numéro et identificateur)
- Programme : via instruction RAISE en fonction des conditions

Exemple **:**

- Soit aucune erreur ne se produit => bloc exception ignoré et exécution du traitement.
- Soit une anomalie se produit => EXECUTION du bloc EXCEPTION avec prise en compte de l'erreur dans les WHEN ou alors traitement dans la partie OTHERS (instructions s'exécutant). Dans le cas ou il n'y a pas une section OTHERS l'exception sera propagée au programme.



Il existe trois types d'exceptions :

- **Exception interne prédéfinie**
- **Exception utilisateur**
- **Exception interne prédéfinie**

- **Exception interne prédéfinie**

Elles correspondent aux erreurs qui se produisent le plus souvent => ORACLE affecte un nom pour les traiter dans la partie EXCEPTION

Nom de l'exception	Numéro	Commentaires
ACCESS_INTO_NULL	ORA-06530	Affectation d'une valeur à un objet non initialisé.
CASE_NOT_FOUND	ORA-06592	Aucun des choix de la structure CASE sans ELSE n'est effectué.
CURSOR_ALREADY_OPEN	ORA-06511	Ouverture d'un curseur déjà ouvert.
DUP_VAL_ON_INDEX	ORA-00001	Insertion d'une ligne en doublon (clé primaire).
INVALID_CURSOR	ORA-01001	Ouverture interdite sur un curseur.
INVALID_NUMBER	ORA-01722	Échec d'une conversion d'une chaîne de caractères en NUMBER.
NO_DATA_FOUND	ORA-01403	Requête ne retournant aucun résultat.
PROGRAM_ERROR	ORA-06501	Problème PL/SQL interne (invitation au contact du support...).
ROWTYPE_MISMATCH	ORA-06504	Incompatibilité de types entre une variable externe et une variable PL/SQL.
STORAGE_ERROR	ORA-06500	Dépassement de capacité mémoire.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Référence à un indice incorrect d'une collection (*nested table* ou *varray*) ou variables de type TABLE.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	
TIMEOUT_ON_RESOURCE	ORA-00051	Dépassement du délai alloué à une ressource.
TOO_MANY_ROWS	ORA-01422	Requête retournant plusieurs lignes.
VALUE_ERROR	ORA-06502	Erreur arithmétique (conversion, troncature, taille) d'un NUMBER.
ZERO_DIVIDE	ORA-01476	Division par zéro.

- **Exception utilisateur**

Définition : Il est possible de définir ses propres exceptions => blocs de traitement des erreurs.

Etape d'exécution d'une exception utilisateur :

Déclaration : déclaration du nom de l'exception dans la partie déclarative du sous-programme

Nomexception EXCEPTION

Déclenchement : le programme va dérouter le traitement vers le bloc des exceptions via la directive **RAISE**

- **Exception interne prédéfinie**

Définition : Utilisation de la directive PRAGME EXECPTION_INIT pour associer un nom d'exception prédéfini à un code d'erreur ORACLE

Etape d'exécution d'une exception interne prédéfinie :

Déclaration : Il faut utiliser deux commandes dans la section déclarative

Nomexception EXCEPTION

PRAGMA EXCEPTION_INIT (nomexception,numéroerreuroracle) ;

Déclenchement : son mode de fonctionnement est identique à une exception prédéfinie. Le servuer SQL aura renvoyé une suite à une instruction SQL

Remarque : Il est possible que le bloc EXCEPTION ne traite pas correctement une exception car il n'existe pas une entrée de bloc correspondante à l'exception et pas de OTHERS => L'exception se propage alors successivement au niveau des blocs EXCEPTION. Si aucun des blocs d'erreur ne peut traiter l'exception le programme principal se termine anormalement.