PL/ SQL

- Introduction:
  - PL/SQL (Procedural Language SQL) est un langage procedural structuré en BLOCS.
  - Extension du SQL: des requêtes SQL intégrées avec les structures de contrôle habituelles (alternatives, répétitives)
  - Un programme PL/SQL est composé de fonctions, de procédures, de triggers..
  - C'est un langage propriétaire d'ORACLE.
  - Peut s'exécuter comme bloc anonyme, procédure ou une fonction

PL/ SQL

- Les blocs PL/SQL: Un programme PL/SQL est constitué de trois blocs ou de trois sections.
  - La section déclarative (optionnelle)
  - La section de contrôle ou d'exécution (obligatoire)
  - La section de gestion des exception (optionnelle)
- La section déclarative: dans cette section, on déclare toutes les variables nécessaires à l'exécution du programme PL/SQL. Cette section commence en général par le mot réservé : DECLARE
- La section d'exécution: cette section contient des énoncés SQL ou PL/SQL. Elle débute par le mot réservé BEGIN et se termine par le mot réservé END.
- La section de gestion des exceptions: cette section commence par le mot réservé EXCEPTION. Si un erreur est générée lors de l'exécution d'un programme , celle-ci est envoyée au BLOC EXECPTION, ce qui donne la possibilité de la traiter et de ne pas mettre fin brutalement à l'exécution du Programme.

PL/ SQL

- Exemple de déclarations:

```
DECLARE
Numero NUMBER(4);
nom VARCHAR2(30);
Salaire NUMBER(8,2);
Date_naissance DATE;
// assignation de valeur
Augmentatation NUMBER (4) := 5;
Groupe VARCHAR2(10) := 'groupe1';

// utilisation du %TYPE. Permet de déclarer des variable de même type
    que des variables déjà déclarée.
Numero_Client  yacoubsa.clients.numclient%type
permet de déclarer le numéro du client du même type que le numclient
    de la table clients de l'usager yacoubsa.
```

PL/ SQL

- Suite Exemple de déclarations:

  Salaire_MIN NUMBER(7,2);

  Salaire_MAX Salaire_Min%TYPE;

  Acteur VARCHAR2(30);

  Realisateur Acteur%TYPE :='Spielberg';

  // on peut donner le type ROWTYPE lorsqu'il s'agit de déclarer un enregistrement (une ligne).

  Enregistrement etudiant%ROWTYPE

Type de données:

- Les types SQL (number, varchar2(n), date, char(n), ..) http://download.oracle.com/docs/cd/E11882_01/appdev.112/e17126.pdf (page 110)
- Le type Boolean
- Le type variable %TYPE
- Le type rangée %ROWTYPE
- Les types CURSOR. (dynamique ou non)
- Les types RECORD.

EXEMPLE: TYPE RECORD

```
SET SERVEROUTPUT ON;
DECLARE
TYPE ENRE IS RECORD
(
Nom1 USER1.ETUDIANTS.NOM%TYPE,
prenom1 VARCHAR2(20)
);
numad1 NUMBER:=20;
ENR ENRE;
BEGIN
SELECT  nom, prenom INTO ENR.nom1,ENR.prenom1
FROM  etudiants
WHERE numad = numad1;
DBMS_OUTPUT.PUT_LINE('le nom est '|| ENR.nom1 || 'le prenom est '
    ||ENR.prenom1);
END;
```

Le type CURSOR

- Un curseur est une zone de mémoire utilisée par Oracle pour récupérer les résultats de requêtes SQL.
- Il peut être explicite, il est donc associé à une seule requête
  - Exemple: CURSOR Resultat IS select nom, preneom from etudiant where nom like %POIT%;
- Il peut être dynamique, n'est pas associé à une seule requête SQL. Dans ce cas, lors de sa déclaration, il faudra utiliser le mot réservé REF.
  - Exemple: TYPE enregistrementEtudiant is REF CURSOR (la variable enregistrementEtudiant est de type CURSOR dynamique).
  - Le type CURSOR sera détaillé plus loin

EXEMPLE: TYPE CURSOR

```
SET SERVEROUTPUT ON;
DECLARE
CURSOR CURSEUR1 IS SELECT * FROM etudiants;
ligne CURSEUR1%rowtype;
BEGIN
OPEN CURSEUR1;
LOOP
FETCH CURSEUR1 INTO ligne;
DBMS_OUTPUT.PUT_LINE('le nom est '|| ligne.nom || 'le prenom est '||
    LIGNE.prenom);
EXIT WHEN curseur1%NOTFOUND;
END LOOP;
CLOSE CURSEUR1;
END;
```

Les instructions de contrôl
Instruction IF –END IF

- **IF - THEN**

  IF condition THEN
  Séquence_instructions;
  END IF;

- **IF - THEN - ELSE**

  IF condition THEN
  Séquence_instructions1;
  ELSE
  Séquence_instructions2;
  END IF;

Instruction IF –END IF

- **IF- THEN - ELSIF**

```
IF condition THEN
        Séquence_instructions1;
ELSIF condition THEN
        Séquence_instructions2;
ELSIF condition THEN
        Séquence_instructions3;
ELSIF condition THEN
        Séquence_instructions4;
ELSE
        Séquence_instructions5;
END IF;
```

Exemple

```
Declare choix number;
begin
  IF choix =1 THEN
     delete from commander where numarticle = 100 ;
  ELSIF choix =2 THEN
     delete from commander where numarticle = 110;
 ELSE
      delete from commander where numarticle = 130;
  END IF;
END;
```

IF – END IF

- Dans une alternative, les mot réservés IF, THEN et END IF sont obligatoires. Les autres (ELSIF et ELSE)sont optionnels
- Exercice: écrire un bloc PL/SQL qui permet de déclarer deux variable de type number (vente et bonus) et qui met à jour le salaire de l'employé comme suit: (salaire = salaire+bonus)
  - Si vente est > 1000 alors bonus = vente*50%
  - Sinon bonus = vente *20%

Réponse

```
CREATE OR REPLACE PROCEDURE Augmentation (vente in
    number) AS
Bonus number;


BEGIN
IF VENTE > 1000 THEN BONUS := VENTE*0.5;
ELSE
BONUS := VENTE*0.2;
END IF;
UPDATE EMPLOYES SET SALAIRE = SALAIRE+BONUS;
Commit;
END;
```

CASE --- WHEN

- L'instruction CASE: permet d'exécuter un bloc PL/SQL selon la valeur d'une variable
- Exemple:

```
CREATE OR REPLACE PROCEDURE CASE1(CHOIX IN NUMBER) AS
BEGIN
CASE CHOIX
WHEN 1 THEN INSERT INTO  employes (numemp, salaire )VALUES (44,28000);
WHEN 2 THEN UPDATE employes SET salaire = salaire +10 where
    numemp =10;
ELSE dbms_output.put_line('pas bon choix');
END CASE;
END;
```

LOOP

- Permet d'exécuter une boucle

Loop
Sequence_instructions
End loop
EXIT WHEN permet de sortir de la boucle.
Exemple:

```
CREATE OR REPLACE FUNCTION COMPTER RETURN NUMBER AS
compteur number:=0;
BEGIN
LOOP
compteur:=compteur+1;
EXIT  WHEN  compteur=10;
END LOOP ;
RETURN compteur;
END;
```

Loop avec EXIT

- Lors de l'execution d'un loop, on peut décider de sortir immediatement de la boucle avec la clause EXIT
- Exemple

```
DECLARE
Credit NUMBER := 0;
BEGIN
    LOOP
Credit := Credit  + 1;
            IF Credit> 3 THEN
            EXIT; -- on sort du loop
            END IF;
    END LOOP;
-- pour afficher le résultat du loop tout de suite
DBMS_OUTPUT.PUT_LINE ('Credit : ' || TO_CHAR(Credit));
END;
```

FOR LOOP

```
set serveroutput on;
BEGIN
   FOR i IN  1..3 LOOP
  DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
   END LOOP;
END;
```

WHILE LOOP

```
 set serveroutput on;
DECLARE
I NUMBER:=1;
BEGIN
   WHILE I < 10 LOOP
   I:= I+1;
   DBMS_OUTPUT.PUT_LINE (TO_CHAR(I));
   END LOOP;
END;
```

Décrémentation

Exemple

```
set serveroutput on;
BEGIN
   FOR i IN REVERSE  1..3 LOOP
   DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
   END LOOP;
END;
```

Voir le site pour les détails
http://docs.oracle.com/cd/B10500_01/appdev.920/a96624/04_struc.htm

Cursor : statique (explicite)

- Un Curseur statique ou explicite est obtenu par l'exécution d'une commande SQL.
- Pour son utilisation, il faut quatre étapes:
  - Déclaration du curseur
  - Ouverture du curseur OPEN
  - Lecture du curseur FETCH
  - Fermeture du curseur CLOSE
- Exemple:

Cursor : statique (explicite)

```
SET SERVEROUTPUT ON;
DECLARE
Nom1 USER1.ETUDIANTS.NOM%TYPE;Prenom1 varchar2(20);
CURSOR curseur1 IS SELECT nom, Prenom from etudiants;
BEGIN
OPEN curseur1;
LOOP
FETCH curseur1 INTO Nom1, Prenom1;
EXIT WHEN curseur1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('le nom est '|| nom1 || 'le prenom est '||
    prenom1);
END LOOP;
CLOSE Curseur1;
END;
```

Cursor : statique (explicite)

Explications:

Les variables nom1 et prenom1 sont des variables locales destinées à recevoir le contenu du curseur.

La commande FETCH permet de lire ligne par ligne le contenu du curseur. À chaque fois que cette commande est appelée, le curseur avance au prochain enregistrement dans l'ensemble actif.

Curseur1%notfound : retourne vrai si le dernier FETCH échoue et ne retourne aucun enregistrement. %notfound est un attribut du curseur explicite.

Cursor : statique (explicite)

Autres attributs du curseur explicite:
% found: contraire de %notfound:
Exemple:

```
CREATE OR REPLACE PROCEDURE test1insertion (NEMP IN VARCHAR2, PEMP IN VARCHAR2)AS
CURSOR CURSEUR1 IS SELECT * FROM TEST1;
NOMEMP VARCHAR2(30);
PRN VARCHAR2(30);
BEGIN
OPEN CURSEUR1;
LOOP
FETCH CURSEUR1 INTO NOMEMP,PRN;
IF CURSEUR1%FOUND THEN
INSERT INTO TEST1 VALUES(NEMP,PEMP);
COMMIT;
ELSE EXIT;
END IF;
END LOOP;
CLOSE CURSEUR1;
  end test1insertion;
```

Cursor : statique (explicite)

Autres attributs du curseur explicite:

%rowcount: retourne le nombre d'enregistrements trouvés

Exemple:

```
LOOP
    FETCH emp_curseur INTO
            emp_nom, dept_num
IF emp_cursor%rowcount>10
THEN EXIT;
END IF;
END LOOP;
```

Cursor : statique (explicite)

Autres attributs du curseur explicite:

%isopen: retourne vrai si le curseur est ouvert:

Exemple:

```
IF emp_curseur%isopen THEN
  FETCH ...
ELSE
    OPEN emp_curseur
END IF;
```

OUVRIR UN CURSEUR POUR MODIFIER UNE COLONNE

```
DECLARE CURSOR CUR1 FOR UPDATE
BEGIN
OPEN CUR1
LOOP FETCH …
IF…
THEN UPDATE … WHERE CURRENT OF CUR1;
END IF;
END LOOP;
CLOSE CUR1;
END;
```

EXERCICE

- ÉCRIRE UNE PROCÉDURE QUI:
- LIT LES COLONNES VENTES DE LA TABLES EMPLOYÉS ET MET À JOUR LA COLONNE COMMISSION COMME SUIT:
- SI VENTE > =2000 ALORS LA COMMISSION EST VENTES*0.5. SINON COMMISSION EST

VENTES *0.1

RÉPONSE:

```
CREATE OR REPLACE PROCEDURE UPDATEEEMPLOYE AS
vente number;
CURSOR  cur1 IS SELECT ventes from employes  for update of
    commision;
BEGIN
 open cur1;
LOOP
fetch cur1 into vente;
if cur1%notfound then exit; end if;
IF VENTE >=2000 THEN update employes set commision =
    vente*0.5 where current of cur1;
Commit;
```

RÉPONSE:

```
ELSE
update employes set commision = vente*0.1 where
    current of cur1;
 commit;


END IF;
END LOOP;
close cur1;


END UPDATEEEMPLOYE;
```