# PL/SQL : Procedural Language/Structured Query Language

D'après le polycopié de Michel Da Paz.

## 1 Exemple de bloc PL/SQL

```
/*-----BLOC AUGMENTATION----*/
DECLARE -- **** Section facultative, déclaration des variables. ****
fonction                        personnel.qualif%TYPE;
traitement_moyen                NUMBER(8,2);
traitement                      traitement_moyen%TYPE;
traitement_null                 EXCEPTION;

BEGIN -- **** Section obligatoire, traitement. ****
SELECT qualif, trtmnt
INTO fonction, traitement
FROM personnel
WHERE nom = '&nom' AND prnm = '&prenom';

SELECT ROUND( AVG(trtmnt), 2 )
INTO traitement_moyen
FROM personnel
WHERE qualif = fonction;

IF traitement IS NULL
THEN RAISE traitement_null;
ELSIF traitement >= traitement_moyen
THEN traitement := ROUND( traitement * 1.01, 2);
ELSE traitement := traitement_moyen;
END IF;

UPDATE personnel
SET trtmnt = traitement
WHERE nom = '&nom' AND prnm = '&prenom';

EXCEPTION -- **** Section facultative, gestion des anomalies. ****
WHEN traitement_null
THEN INSERT INTO anomalie
VALUES('&nom', '&prenom', '!!!! TRAITEMENT NON DEFINI !!!!!');
END;
/
```

# 2 Structure d'un bloc PL/SQL

Un bloc comporte de 1 à 3 sections.

```
DECLARE  Section facultative
Déclaration des variables et constantes

BEGIN  Section obligatoire
Traitement ( instructions SQL et PL /SQL )
Imbrication de blocs possible

EXCEPTION Section facultative
Gestion des erreurs

END;
/ -- Exécution du bloc
```

## 2.1 Commentaire

Il y a deux formes de commentaire.
syntaxe langage C : `/* commentaire multilignes */`
début : `-- commentaire uniligne`

## 2.2 Portée des variables

La portée des variables suit les règles du langage C ( imbrication de blocs ).

```
<<b_srv>>
DECLARE
  nom VARCHAR(25);
BEGIN
  ...
  DECLARE
    nom VARCHAR(25);
  BEGIN
    ...
    IF nom = b_srv.nom
    ...
    END IF;
  END;
END b_srv;
```

# 3 Déclarations

Section facultative.

- Variables et constantes : les types scalaires sont:
  - nombres :
    - BINARY_INTEGER (entiers signés)
    - NUMBER[(precision,scale)] , précision = nombre totale de chiffres et scale= nombre de chiffres après la virgule. (NUMBER(precision)≡ NUMBER(precision,0))
  - caractères :
    - CHAR[(longueurmax)] : chaine de caractères de longueur fixe par défaut longueur max=1
    - VARCHAR2(longueurmax), LONG : chaînes de longueur variable
    - RAW, LONG RAW : chaînes de bits
    - NCHAR, NVARCHAR2 : chaîne de caractères NLS (permet de coder les caractères sur 8 bits : caractères accentués)
    - ROWID : chaîne hexadécimale servant à coder l'adresse d'un tuple dans une table.
  - BOOLEAN
  - DATE
- Curseurs
- Pointeurs REF CURSOR, REF <type_objet>
- types de gros objets BFILE, BLOB, CLOB, NCLOB
- Types composés (RECORD, TABLE et VARRAY)
  - Enregistrement (TYPE RECORD) : Un RECORD doit être déclaré en deux étapes :
    - Déclaration du type d'enregistrement :

```
TYPE nom_type IS RECORD
(nom_champ {type_champ|variable%TYPE|
            table.colonne%TYPE|table%ROWTYPE} [ NOT NULL ],
... );
```

Ex :
```
TYPE R_emp Is RECORD
( nom VARCHAR(25) NOT NULL,
  datnais DATE,
  qualif personnel.qualif%TYPE );
```

* Déclarations des enregistrements de ce type :
  `variable   nom_type;`
  Ex : `v_emp   R_emp;`
- TYPE TABLE : Une table PL/SQL se comporte comme un tableau. Une TABLE doit être déclarée en deux étapes :
  * Déclaration du type de la table :
    ```
    TYPE nom_type IS TABLE OF
    { type_colonne | variable%TYPE | table.colonne%TYPE} [ NOT NULL]
    INDEX BY BINARY_INTEGER;
    ```
    Ex :
    ```
    TYPE T_nom IS TABLE OF
         personnel.nom%TYPE
    INDEX BY BINARY_INTEGER;
    ```
  * Déclarations des tables PL/SQL de ce type :
    `variable   nom_type;`
    Ex : `v_nom   T_nom;`

- Exceptions

# 4 Corps du programme

Séquence d'instructions SQL et PL/SQL.

## 4.1 Affectation des variables

Deux possibilités

- Opérateur d'affectation `:=`
- Ordre `SELECT ... INTO`

## 4.2 Traitement conditionnel

Syntaxe:

```
IF     condition
       THEN .....
       [ELSE] [ELSIF] ......
END IF;
```

Ex: `IF traitement IS NULL`

```
      THEN RAISE traitement_null;
      ELSIF traitement >= traitement_moyen
            THEN traitement := ROUND(traitement * 1.01, 2);
            ELSE traitement := traitement_moyen;
  END IF;
```

## 4.3 Branchement inconditionnel

Syntaxe :

```
<<étiquette>>                                GOTO étiquette
..........                                     .......
......                       ou                ....
GOTO étiquette                               <<étiquette>>
```

## 4.4 Instruction NULL

Aucune action n'est effectuée.
Ex : `IF traitement IS NULL`

```
      THEN NULL;
      ELSIF traitement >= traitement_moyen
            THEN traitement := ROUND(traitement * 1-m, 2);
            ELSE traitement := traitement_moyen;
   END IF;
```

## 4.5 Traitement itératif

### 4.5.1 LOOP

Permet d'exécuter des boucles infinies.
syntaxe: `[<<étiquette>>]`

```
       LOOP
       ......
       END LOOP [étiquette];
```

La commande EXIT permet de sortir d'une boucle sans condition.
La commande EXIT WHEN <condition> permet de sortir d'une boucle si la condition est vraie.
Ex: `DECLARE`

```
       i BINARY_INTEGER:=1;
    BEGIN
      LOOP
```

```
    EXIT WHEN i>10;
    INSERT INTO l10nombres VALUES ('le nombre est ',i);
    i:=i+1;
   END LOOP;
 END;
```

### 4.5.2 WHILE LOOP

Boucle exécutée tant qu'une condition est vraie.

```
[<<étiquette>>]
WHILE condition
LOOP

..... .
END LOOP [étiquette];
```

Ex:DECLARE

```
   i BYNARY_INTEGER:=1;
 BEGIN
   WHILE i <= 10
   LOOP
    INSERT INTO l10nombres VALUES ('le nombre est ',i);
    i:=i+1;
   END LOOP;
 END;
```

### 4.5.3 FOR LOOP

Permet de limiter le nombre d'itérations de la boude et de disposer du numéro d'itération.
Syntaxe : `[<<étiquette>>]`

```
FOR indice IN [REVERSE] indice_début .. indice_fin
......
END LOOP [étiquette];
```

Ex: DECLARE

```
BEGIN
 FOR i IN 1..10
 LOOP
    INSERT INTO l10nombres VALUES ('le nombre est',i);
 END LOOP;
END;
```

### 4.6 Utilisation des enregistrements

Ex : DECLARE

```
 TYPE R_emp IS RECORD( nom VARCHAR(25) NOT NULL,
                       datnais DATE,
                       qualif personnel.qualif%TYPE );
 v_emp      R_emp;
BEGIN
 v_emp.datnais := '24-MAR-38';
 SELECT nom, datnais, qualif
        INTO v_emp
        FROM personnel
        WHERE nom||prnm ='&nom'||'&prnm';
 ......
END;
```

### 4.7 Utilisation des tables

Ex : DECLARE

```
 TYPE T_nom IS TABLE OF personnel.nom%TYPE
      INDEX BY BINARY_INTEGER;
 v_nom      T_nom;
 i          BINARY_INTEGER := 0;

BEGIN
 FOR r_emp IN ( SELECT nom, trtmnt FROM personnel ORDER BY trtmnt )
 LOOP
     i := i+ 1;
     v_nom ( i ) := r_emp.nom;
 END LOOP;
 proc_trtmnt ( v_nom );
END.
```

## 5 Curseur :

Un curseur est un outil qui permet de parcourir le résultat d'une requête, tuple par tuple.

### 5.1 Déclaration du curseur

Ex : `CURSOR c_traitement IS`

```
SELECT nom, prnm, qualit, dattit, trtmnt
FROM personnel
ORDER BY 5;
```

## 5.2 Ouverture du curseur

Ex : `OPEN c_traitement;`

## 5.3 Traitement des lignes

Ordre FETCH utilisable sur des variables indépendantes ou sur un enregistrement.

### 5.3.1 Utilisation de variables indépendantes

Ex : `DECLARE`

```
 CURSOR c_fonction IS
      SELECT nom, prnm, sx, qualif, DECODE(qualif,'AIDE SOIGNANTE','1',
             'INFIRMIERE','2', 'INTERNE','3','9')
      FROM personnel
      ORDER BY 5;

 nom                personnel.nom%TYPE;
 prenom             personnel.prnm%TYPE;
 sexe               personnel.sx%TYPE;
 fonction           personnel.qualif% TYPE;
 cdqualif           CHAR(1);
BEGIN
...FETCH c_fonction INTO nom, prenom, sexe, fonction, cdqualif;
END;
```

### 5.3.2 Utilisation d'un enregistrement

Ex : `CURSOR c_traitement IS`

```
     SELECT nom, pmm, qualif, dattit, trtmnt
     FROM personnel
     ORDER BY 5;
 r_traitement c_traitement%ROWTYPE;
 ...
 FETCH c_traitement INTO r_traitement;
```

## 5.4 Tests sur un curseur

- `nom_curseur%ISOPEN` : TRUE si curseur ouvert, FALSE si curseur fermé.
- `nom_curseur%ROWCOUNT` : Indique le nombre de lignes déjà retournées par l'ordre FETCH.
- `nom_curseur%FOUND` : TRUE si le dernier FETCH a ramené une ligne, FALSE sinon.
- `nom_curseur%NOTFOUND` : TRUE si le dernier FETCH n'a pas ramené de ligne ( fin du curseur ), FALSE sinon.

## 5.5 Exemples sur les curseurs

Ex1:
```
DECLARE

 CURSOR c_traitement IS
      SELECT nom, prnm, qualif, dattit, trtmnt
      FROM personnel
      ORDER BY 5;

  r_traitement c_traitement%ROWTYPE;
BEGIN
  OPEN c_traitement;
  LOOP
   FETCH c_traitement INTO r_traitement;
   EXIT WHEN c_traitement%NOTFOUND OR c_traitement%ROWCOUNT = 6;
   INSERT INTO traitement VALUES (r_traitement.nom, r_traitement.prnm,
         r_traitement.qualif,r_traitement.dattit,r_traitement.trtmnt );
  END LOOP;
  CLOSE c_traitement;
END;
```

Ex2:
```
DECLARE

  CURSOR c_titulaire IS
         SELECT nom, prnm, datnais, qualif, dattit
         FROM personnel
         WHERE dattit IS NOT NULL
         ORDER BY dattit - datnais ;

  nom      personnel.nom%TYPE;
  prenom   personnel.prnm%TYPE;
  datnais  personnel.datnais%TYPE;
  fonction personnel.qualif%TYPE;
```

```
 dattit   personnel.dattit%TYPE;
 agetit   NUMBER(2) := 0;
 P       NUMBER(2) := 0;

BEGIN
 OPEN c_titulaire;
 WHILE p < 8 AND agetit < 25
 LOOP
    FETCH c_titulaire INTO nom, prenom, datnais, fonction, dattit;
    EXIT WHEN c_titulaire%NOTFOUND;
    p := p +1;
    agetit := ROUND((dattit - datnais) / 365.25);
    INSERT INTO titulaire VALUES(p,nom,prenom,datnais,fonction,dattit,agetit);
 END LOOP;
 CLOSE c_titulaire;
END;
```

Ex3:
```
DECLARE

 CURSOR c_anciennete IS
        SELECT nom, prnm, qualif, dattit, trtmnt
        FROM personnel
        WHERE trtmnt IS NOT NULL
        AND dattit IS NOT NULL
        ORDER BY dattit DESC ;

 nom         personnel.nom%TYPE;
 prenom      personnel.prnm%TYPE;
 datnais     personnel.datnais%TYPE;
 fonction    personnel.qualif%TYPE;
 dattit      personnel.dattit%TYPE;
 traitement  personneI.trtmnt%TYPE;
 anciennete  NUMBER(2);

BEGIN
 OPEN c_anciennete;
 FOR p IN 1..&nbprs
 LOOP
     FETCH c_anciennete INTO nom, prenom, fonction, dattit, traitement ;
     EXIT WHEN c_anciennete%NOTFOUND;
     anciennete := ROUND((SYSDATE - dattit) / 365.25);
     INSERT INTO anciennete VALUES(p,nom,prenom,fonction,anciennete,traitement ) ;
 END LOOP;
 CLOSE c_anciennete;
```

```
END;
```

### 5.6 Fermeture d'un curseur

Libération de la place mémoire.
Ex : `CLOSE c_traitement;`

### 5.7 Boucles FOR LOOP sur un curseur

Les ordres OPEN, FETCH et CLOSE sont implicites.
syntaxe `[<<étiquette>>]`

```
    FOR nom_enregistrement IN nom_curseur | (SELECT... )
    LOOP
       ....
    END LOOP [étiquette];
```

Ex 1: `DECLARE`

```
  CURSOR c_docteur IS
         SELECT s.nom serv, p.nom nom, prnm, vacct
         FROM service s, personnel p
         WHERE s.cdsrv = p.cdsrv
         AND qualif = 'DOCTEUR' .
         AND vacct < (SELECT AVG(trtmnt) * 0.75
         FROM personnel
         WHERE qualif = 'INTERNE'
         AND cdsrv = p.cdsrv);

 BEGIN
  FOR I IN c_docteur
  LOOP
       INSERT INTO docteur VALUES ( I.serv,I.nom,I.prnm,I.vacct ) ;
  END LOOP;
 END;
```

Ex2: `BEGIN`

```
  FOR I IN (SELECT nom, prnm, sx, qualif, dattit
            FROM personnel
            WHERE sx = '2' AND TO_CHAR(dattit,'YY') IN ('90','91'))
  LOOP
   INSERT INTO embauche VALUES ( I.nom,I.prnm,I.sx,I.qualif,I.dattit );
  END LOOP;
 END;
```

### 5.8 Curseur paramétré

Syntaxe: `CURSOR nom_curseur ( param_1 type_1, param_2 type_2, ... ) IS`

```
                 SELECT ....
        OPEN nom_curseur ( valeur_1, valeur_2, ... );
```

Ex : `DECLARE`

```
     CURSOR c_personne (tranche NUMBER) IS
            SELECT nom, prnm, trtmnt
            FROM personnel
            WHERE TRUNC(trtmnt,-3) > tranche;

    BEGIN
     FOR p IN c_personne (15000)
     LOOP
         INSERT INTO traitement VALUES( p.nom,p.prnm,p.trtmnt ) ;
     END LOOP;
    END;
```

## 6 Gestion des exceptions

Cette section est facultative.

### 6.1 Anomalie utilisateur

```
DECLARE
 nom_anomalie     EXCEPTION;     -- Nomme l'anomalie.
BEGIN
 IF condition                    -- Détecte l'anomalie.
    THEN RAISE nom_anomalie;     -- Transfère le contrôle au traitement de l'anomalie.
 ......
EXCEPTION
 WHEN nom_anomalie               -- Traitement de l'anomalie puis
      THEN ....                  -- passage à l'environnement hôte.
END;
```

Ex: `DECLARE`

```
    e_insee EXCEPTION;
   BEGIN
    IF LENGTH( '&insee' ) != 13
       THEN RAISE e_insee;
```

```
    ELSIF ......
  END IF;
 EXCEPTION
  WHEN e_insee
       THEN INSERT INTO anomalie VALUES ( '&nom'||' '||'!!!!! INSEE !!!!!');
  WHEN ......
 END;
```

Le même traitement peut être appliqué à plusieurs exceptions :

```
WHEN e_1 OR e_2 OR e_3
     THEN ...
```

## 6.2 Erreur ORACLE prédéfinie

Syntaxe : `EXCEPTION`

```
          WHEN nom_erreur_prédéfinie
               THEN ......
```

Ex : `BEGIN`

```
     INSERT INTO malade VALUES ( '&nom', '&datnais', '&insee'); -- Index primaire
    EXCEPTION
     WHEN DUP_VAL_ON_INDEX
          THEN INSERT INTO anomalie VALLUES('&nom'||' '||'!!!! INSEE DUPLIQUE!!!!!');
```

## 6.3 Erreur nommée

Syntaxe: `DECLARE`

```
           nom_erreur EXCEPTION;
           PRAGMA EXCEPTION_INIT (nom_erreur, code_erreur_ORACLE);
         EXCEPTION
           WHEN nom_erreur THEN ......
         END;
```

Ex : `DECLARE`

```
         e_dater EXCEPTION;
         PRAGMA EXCEPTION_INIT (e_dater, -01847 );
     BEGIN
         v_dat := TO_DATE ( '&datref');
     EXCEPTION
         WHEN e_dater THEN ......
     END;
```

## 6.4 Exceptions internes génériques

Utilisation de la condition OTHERS.

```
EXCEPTION
  WHEN nom_erreur_1
  WHEN nom_erreur_2
  WHEN OTHERS
       THEN ......
END;
```

## 6.5 Propagation d'une exception

```
BEGIN
  BEGIN
    IF x =1                         e_1
       THEN RAISE e_1 ;             ---
       ELSIF x = 2                    |        e_2
             THEN RAISE e_2 ;         |       ----      e_3
             ELSE RAISE e_3 ;         |          |     ----
    END IF ;                          |          |        |
    ...                               |          |        |
  EXCEPTION      <------------------  <--------  <------
    WHEN e_1
         THEN ...    ----------------
  END;                                |   --------  -------
EXCEPTION      <---------------------          |  <-----|
  WHEN e_2                                     |
       THEN ...                           <-------
  ...          ---------------------   --------  -------
  ...                                 |          |        |
END ;          <---------------------   <-------  <------
```

## 6.6 Continuation après traitement d'une exception

Ex : `BEGIN`

```
        SQL1
        BEGIN
           SQL2
           ...
        EXCEPTION
           WHEN ...     -- Traitement de l'exception pour SQL2
        END ;
```

```
SQL3            -- Continuation de l'exécution du bloc principal
...
```

# 7 Sous programmes : procédures, fonctions

## 7.1 Création d'une procédure, fonction

```
CREATE PROCEDURE p_augmnt
( a_pourc  IN NUMBER,
  a_qualif IN VARCHAR ) IS
BEGIN
  UPDATE personnel
     SET trtmnt = ROUND ( trtmnt * ( 1 + a_pourc / 100 ), 2 )
     WHERE qualif = a_qualif;
  IF SQL%NOT_FOUND '
     THEN RAISE_AFPLICATION_ERROR ( -20011, 'qualification inexistante' );
  ENDIF;
END ;

CREATE OR REPLACE FUNCTION f_salaire
( a_nom IN VARCHAR,
  a_prnm IN VARCHAR )
RETURN NUMBER IS
  a_renum NUMBER(8,2);
BEGIN
  SELECT NVL(trtmnt, 0) + NVL(vact, 0)
         INTO a_renum
         FROM personnel
         WHERE nom||prnm = a_nom||a_prnm;
  RETURN (a_renum);
EXCEPTION
  WHEN NO_DATA_FOUND
       THEN RAISE_APPUCATION_ERROR(-20010, 'Personnel inexistant');
END;
```

## 7.2 Modification d'une procédure, fonction

```
ALTER PROCEDURE p_augmnt COMPILE;
```

## 7.3 Suppression d'une procédure fonction

```
DROP PROCEDURE p_augmnt;
```

# 8 Paquetages

C'est un ensemble de procédures et de fonctions.

## 8.1 Création d'un paquetage

### 8.1.1 Création des spécifications du paquetage

```
CREATE PACKAGE ges_pers IS
       FUNCTION f_age ( a_nom VARCHAR, a_prnm VARCHAR )
                RETURN NUMBER
END;
```

### 8.1.2 Création du corps du paquetage

```
CREATE PACKAGE BODY ges_pers IS
      FUNCTION f_age ( a_nom VARCHAR, a_prnm VARCHAR )
         RETURN a_age NUMBER IS
         BEGIN
           SELECT ROUND ( (SYSDATE - datnais) / 365.25 )
                  INTO a_age
                  FROM personnel
                  WHERE nom||prnm = a_nom||a_prnm;
           RETURN ( a_age );
         END;

      PROCEDURE p_suppers ( a_nom VARCHAR, a_pmm VARCHAR ) IS
         BEGIN
           DELETE FROM personnel
                  WHERE nom||prnm = a_nom||a_prnm;
         END;
END;
```

## 8.2 Modification d'un paquetage

```
ALTER PACKAGE ges_pers COMPILE BODY;
```

## 8.3 Suppression d'un paquetage

```
DROP PACKAGE ges_pers;
```

# 9 Les Démons (trigger)

## 9.1 Création d'un trigger

Un trigger est formé de trois parties :

- Définition du déclenchement
- Restriction
- Action

Enregistrements anciennes et nouvelles valeurs de la ligne courante : OLD, NEW

```
CREATE TRIGGER t_sup_srv_prs
       BEFORE DELETE ON service
              FOR EACH ROW
       WHEN ( OLD.nom !='CARDIOLOGIE' )
       BEGIN
            DELETE FROM personnel
                   WHERE cdsrv = :OLD.cdsrv;
       END;
```

## 9.2 Modification d'un trigger

```
ALTER TRIGGER t_sup_srv_prs DISABLE;
ALTER TABLE service DISABLE ALL TRIGGERS;
```

## 9.3 Suppression d'un trigger

```
DROP TRIGGER t_sup_srv_prs;
```

# Contents