PL/SQL - ASHRAF KHABAR

- Introduction :
  - PL/SQL is a prucedural language offered by oracle in order to solve some problems which sql can't do it , like some interactions with the database need some conditions ...
  - PL/SQL is a programming language unlike sql it's a query language .
  - PL/SQL gives us the ability to execute the hole code unlike the sql , we need to wait the execution of the first querythen run the next one , for example , we cannot run selection at the same time as a drop , but with PL/SQL we can make a block of instructions to make a lots of queries at the same time .
  - PL/SQL is block language .
  - we have 3 blocks , block of functions , block of Procedure , and blocks Anonymous (without name and not stored in the database => executed at the time of runing).
  - The structure of the PL/SQL :

```
DECLARE
    -- declarations
BEGIN
    -- instructions pl/sql or sql
EXCEPTION
    -- exceptions
END ;
```

- Variables :
  - Type Scalaire : Char, Date, number, boolean, int, float....
  - Type Composé : Enregistrement et Table
- Constant :
  -

```
name_of_constant `constant` type_of_constant := value
```

- Declaration by reference :
  -

```
* name_of_variable1 name_of_variable2 %type
* nom_variable nom_table.nom_colonne %type;
```

```
declare
    Nom char(20) ;
    prenom Nom %type ;
```

- Affectation :
  - name_of_variable := value ;
  - name_of_array(i) := value ;
  - name_of_table.name_of_field := value
- Result of query :
id	name	surname	age
1	ashraf	khabar	22
2	sami	aouad	22

```
declare
    age_of_student int ;
begin
    select AGE into age_of_student from STUDENT where id=1;
    DBMS_OUTPUT.PUT_LINE(age_of_student) ;
end;
/
declare
    name_of_student varchar(255) ;
begin
    select name into name_of_student from STUDENT where id=1 ;
    DBMS_OUTPUT.PUT_LINE(name_of_student) ;
end;
/
```

- regestering :
  - ```
    declare
        eleve STUDENT%rowtype ;
    begin
        select * into eleve from STUDENT where id = 1 ;
        DBMS_OUTPUT.PUT_LINE('le nom est : '||eleve.name||', le prenom
    est : '||eleve.surname||' , l age est : '||eleve.age) ;
    end;
    ```

    OR :

    ```
    declare
        type eleve is record (nom varchar(255) , prenom varchar(255) ,
    age int );
        elv eleve ;
    begin
        elv.nom := 'ashraf' ;
    ```
    ```

```
        DBMS_OUTPUT.PUT_LINE('Hello '||elv.nom) ;
    end;
```

- arrays:
    - ```
      declare
          type tab is table of int index by pls_integer ;
          note tab ;
      begin
          note(0) := 12 ;
          note(1) := 20 ;
          DBMS_OUTPUT.PUT_LINE(note(1)) ;
      end;
      ```

- if/else statement :

```
declare
    a int ;
    b int ;
    c int ;
begin
    a := 2 ; b := 3 ; c := 4 ;
    if a > b then
        c := a + b ;
    else
        c := a - b ;
    end if;

    DBMS_OUTPUT.PUT_LINE('The result value is : '||c) ;
end;
```

- elsif and else if :

```
declare
    a int ;
    b int ;
    c int ;
begin
    a := 2 ; b := 3 ; c := 4 ;
    if a > b then
        c := a + b ;
        else if a = b then
            c := 0 ;
        end if;
    else
        c := a - b ;
```
```

```
        end if;

        DBMS_OUTPUT.PUT_LINE('The result value is : '||c) ;
    end;
```

```
    declare
        a int ;
        b int ;
        c int ;
    begin
        a := 2 ; b := 3 ; c := 4 ;
        if a > b then
        c := a + b ;
        elsif a = b then
            c := 0 ;
        else
        c := a - b ;
        end if;

        DBMS_OUTPUT.PUT_LINE('The result value is : '||c) ;
    end;
```

- Case :

```
  declare
      opp char ;
      number1 int ;
      number2 int ;
      res float ;
  begin
      number1 := 2 ;
      number2 := 3 ;
      opp := '/' ;
      case opp
          when '+' then res := number2 + number1 ;
          when '-' then res := number2 - number1 ;
          when '*' then res := number2 * number1 ;
          when '/' then res := number2 / number1 ;
          else res := 0 ;
      end case ;
      DBMS_OUTPUT.PUT_LINE(number1||' '||opp||' '||number2||' = '||res) ;
  end;
```

- While loop :

```
  declare
     month_val int ;
```

```
begin
    loop
        month_val := daily_value * 31;
        EXIT WHEN month_value > 4000;
    end loop ;
end;
```

- for loop :
    - ```
      for variable in [REVERSE] debut..fin
      loop
          -- instructions
      end loop ;
      ```

```
declare
    i number ;
begin
    for i in 1..5 loop
        DBMS_OUTPUT.PUT_LINE('Numbers : '|| i ) ;
    end loop;
end;
```

- loop :

```
declare
    i number ;
begin
    i := 1 ;
    loop
        DBMS_OUTPUT.PUT_LINE('Nombre : '|| i ) ;
        i := i + 1 ;
        exit when i > 5 ;
    end loop;
end;
```

- Exceptions :

```
declare
    opp char ;
    number1 int ;
    number2 int ;
    res float ;
begin
    number1 := 0 ;
    number2 := 3 ;
```
```

```
        opp := '/' ;
        case opp
            when '+' then res := number2 + number1 ;
            when '-' then res := number2 - number1 ;
            when '*' then res := number2 * number1 ;
            when '/' then res := number2 / number1 ;
            else res := 0 ;
        end case ;
        DBMS_OUTPUT.PUT_LINE(number2||' '||opp||' '||number1||' = '||res) ;

        exception
            when VALUE_ERROR then DBMS_OUTPUT.PUT_LINE('ARITHMETIC ERROR') ;
            when OTHERS then DBMS_OUTPUT.PUT_LINE('ERROR') ;
    end;
```

  - Predefined exceptions :
    - NO_DATA_FOUND --> Quand Select into ne retourne aucune ligne
    - TOO_MANY_ROWS --> Quand Select into retourne plusieurs lignes
    - VALUE_ERROR --> (erreur arithmétique)
    - OTHERS --> Toutes erreurs non interceptées

- Create Exceptions :

```
    declare
        DEVIDING_BY_ZERO EXCEPTION ;
        opp char ;
        number1 int ;
        number2 int ;
        res float ;
    begin
        number1 := 0 ;
        number2 := 3 ;
        opp := '/' ;
        if number1 = 0 then RAISE DEVIDING_BY_ZERO ;
        end if ;
        case opp
            when '+' then res := number2 + number1 ;
            when '-' then res := number2 - number1 ;
            when '*' then res := number2 * number1 ;
            when '/' then res := number2 / number1 ;
            else res := 0 ;
        end case ;
        DBMS_OUTPUT.PUT_LINE(number2||' '||opp||' '||number1||' = '||res) ;

        EXCEPTION
            when DEVIDING_BY_ZERO then DBMS_OUTPUT.PUT_LINE('Divinding by 0 is not
allowed') ;
    end;
```

- Curseur :

Syntaxe 👍

```
declare
    cursor C1 is (select * from STUDENT ) ;
begin
    open C1 ;

    close C1 ;
end;
```

fetch 💯 : fetch is used to put the content of cursor into a variable .

- How to range the curser :

```
declare
    cursor c is select * from STUDENT ;
begin
    --it's not necessary to declare the variable v because it's declared
automatically .
    for v in c loop
        DBMS_OUTPUT.PUT_LINE('name est : '||v.NAME) ;
    end loop;
end;
```

AUTRE METHODE :

```
declare
    cursor c is select * from STUDENT ;
    id STUDENT.ID%type ;
    nom STUDENT.NAME%type ;
    prenom STUDENT.SURNAME%type ;
    age STUDENT.AGE%type ;
begin
    open c ;
        loop
            fetch c into id , nom , prenom , age ;
            exit when c%notfound ;
            DBMS_OUTPUT.PUT_LINE('name est : '||nom) ;
        end loop;
    close c ;
end;
```

- Fonctions and Precedures :
  - Syntax : without param

```
create procedure NOMS is
begin
    declare
        cursor c_STUDENT is select * from STUDENT ;
    begin
        for v_STUDENT in c_STUDENT loop
            DBMS_OUTPUT.PUT_LINE('le noms est : '|| v_STUDENT.NAME ) ;
        end loop;
    end;
end;
```

⚠ : in select statement inside procedure we always need the `into` key word .

- syntax : with params

```
create or replace procedure PRINT_NUMBERS(A in int , B in INT) is
    somme int ;
begin
    somme := A + B ;
    DBMS_OUTPUT.PUT_LINE(somme) ;
end;
/
declare

begin
    PRINT_NUMBERS(12,69) ;
end;
```

- Syntax : Function

```
create or replace function PRINT_NUMBERS_FUNCTION(A in int , B in INT)
    return int
    is
    somme int ;
begin
    somme := A + B ;
    return somme ;
end;
/
declare
    som int ;
begin
    som := PRINT_NUMBERS_FUNCTION(12,69) ;
    DBMS_OUTPUT.PUT_LINE(som) ;
end;
```

big notes :

- impossible de ne pas metre `into` dans la requete `select` dans une procedure , c'est obligatoire .
- la requte `select` retourne une seul ligne , si on veut retourner plusieurs lignes on doit utiliser les `curseurs` .
- la date se declare comme ca :

```
declare
    date_var date ;
begin
    date_vare := to_date('02-JUN-2022','DD-MON-YYYY') ;
end ;
```

- pour avoire le meme type d'une variable on utilise `%type`
- pour avoire une variable qui prend toute la ligne d'une table on utilise `%rowtype` .
- l'instruction case renvoie une valeur qui vaut resultat1 ou resultat2 ou ... ou le resultat par defaut , donc on peut faire ca

```
  val := CASE city
      WHEN 'casablanca' THEN '50'
      WHEN 'RABAT' THEN '30'
      WHEN 'Tanger' THEN '20'
      ELSE '0'
      END CASE ;
```

- Dans les boucle `for` on est pas obliger de declarer la variable de bouclage :

```
declare
begin
    for i in 1..5 --la variable i n'est pas declarée
    loop
        DBMS_OUTPUT.PUT_LINE(12);
    end loop;
end;
```

- un `curseur` est un pointeur vers un resultat d'un requete 👍 on les utilise pour les requetes qui retourne plusieurs lignes .
    - on utilise souvent les `curseurs` dans une boucle `for` qui permet une utilisation implicite des instructions `OPEN` , `FETCH` et `CLOSE` .
    - on sort de la boucle quand l'attribut `NOTFOUND` est vrai .
- on peut pas referencer un variale passée au parametre dans une `function` ou `procedure` :

```
create function somme(number1 IN int , number2 IN int ) return int
is
    somme number1%type ; -- c'est faux
    somme number2%type ; -- c'est faux
    somme int ;          -- c'est vrai
begin
    somme := number1 + number2 ;
    return somme ;
end;
```

- dans les `functions` :
  - `IN` : lecture seul
  - `OUT` : ecriture seul
  - `IN OUT` : lecture et ecriture et on l'utilise pour les passage par reference .
- Une variable `global` est declarée par le prefix "`:`" .
- Pour associer a une `exception` un numero on utilise le mot cle `pragma` :

```
declare
      my_exception EXCEPTION ;
      pragma EXCEPTION_INIT ( my_exception , -20000) ;
      number1 number ;
      number2 number ;
      result number ;
      opp char ;
  begin
      number2 := 0 ;
      number1 := 36 ;
      opp := '/' ;

      case opp
          when '+' then result := number1 + number2 ;
          when '-' then result := number1 - number2 ;
          when '*' then result := number1 * number2 ;
          when '/' then
              if number2 = 0 then
                  raise_application_error(-20000,'la division sur 0 est
impossible') ;
              else
                  result := number1 / number2 ;
              end if;
      end case ;
      DBMS_OUTPUT.PUT_LINE(number1||opp||number2||' = '||result) ;

      EXCEPTION
          when my_exception then
              DBMS_OUTPUT.PUT_LINE('la division sur 0 est impossible') ;
          when others then
```

```
                DBMS_OUTPUT.PUT_LINE('erreur') ;
    end ;
```

- dans les fonction est les procedure dans la signature on declare que `varchar` ou `char` sans specifier la longueur , par contre logique interne on declare la longueur :

```
create or replace function fetch_by_name(id IN char )
return varchar2
is
    temp varchar2(255) ;
begin
    select NOM into temp from CLIENT where CODE=id ;
    return temp ;
end;
```

- Triggers :
  - Syntax :

```
create [or replace] trigger trigger_name
{before | after} {insert [or] | update [or] | delete}
[OF col_name] ON table_name
[for each row]
when (condition)
begin
    -- instructions
end;
```

  - `FOR EACH ROW` :
    - false (statement) => executé une seul fois pour la commande
    - true (row) => executé à chaque ligne concerné
  - si on ajoute une variable on la recupere grace a la variable `:NEW.nom_variable` .
  - si on suprime ou on modifie une variable on la recupere grace a la variable `:OLD.nom_variable` .
  - Si on veut preciser ou la modification (sur quelle colone):
    -

```
create or replace trigger TRG_ELEVE
    befor update of MOYENNE
    on ELEVE
    for each row
    when (NEW.MOYENNE > 12)
    begin
        if updating('MOYENNE') then --si on modify la colonne
```

```
moyenne
                dbms_output.put_line('validé') ;
            end if ;
        end ;
```

exemples of function and triggers :

- ```
    create or replace package MY_VARIABLES_TYPE
as
    type list is table of varchar2(255) index by pls_integer ;
end;

create or replace function get_names
return MY_VARIABLES_TYPE.list
is
    NAME_LIST MY_VARIABLES_TYPE.list ;
    cursor c is select NOM from CLIENT ;
    i int ;
begin
    i := 0 ;
    for v in c loop
        NAME_LIST(i) := v.NOM ;
        i := i+1 ;
    end loop;
    return NAME_LIST ;
end;

/
declare
    names MY_VARIABLES_TYPE.list ;
    i int ;
    nbr int;
begin
    select count(NOM) into nbr from CLIENT ;
    names := GET_NAMES();
    i := 0 ;
    loop
        DBMS_OUTPUT.PUT_LINE(names(i)||',') ;
        i := i + 1 ;
        exit when i = nbr ; -- necaissary
    end loop;
end;
```
```