sumPlsql.sql

```
--Transaction: Propriétés
    ACIDE :
            Atomicité
            Consistance
            Isolation
            Durabilité

✨Atomicité
    /*L'ensemble des opérations d'une transaction apparaît comme une seule
    opération atomique
    Soit toutes les opérations sont validées ou toutes annulées (tout ou rien)
    */

✨Consistance
    /*L'exécution de la transaction fait passer la base de données d'un état
    consistant à un autre état consistant*/

✨Isolation
    /*Chaque transaction est indépendante des autres transactions concurrentes.
    Sérialisation des transactions.
    Les résultats d'une transaction ne sont visibles aux autres transactions
    qu'une fois la transaction validée.
    Les concurrences sont parfaitement contrôlées*/

✨Durabilité
    /*C'est la persistance des mises à jour d'une transaction validée.
    Les effets d'une transaction validée sont durables et permanents, quelques
    soient les problèmes logiciels ou matériels, notamment après la fin de la
    transaction.*/

-- base structure :
   declare

   begin

   exceptions

   end;
   /

-- print :
   dbms_output.put_line('your text ');

-- make the sql plus  can print the result of a put_line function :
   set serveroutput on;

-- delcare a new variable
   declare
   variableName datatype

-- concatination :
   value1 || valu2

```

```
 --* example  1  writing my age and full name on the screen :
      set serveroutput on ;
      declare
      age number default 20;
      fullName varchar(100) default 'ayoub majid ';
      begin
      dbms_output.put_line('the age is : ' || age);
      dbms_output.put_line('the full name is : ' || fullName);
      end;
      /
-- assignment 1 :
   -- write a bolck that outputs 'hello' and save it as test.sql
   -- then run this script again
-- solution :
   set serveroutput on;
      begin
      dbms_output.put_line('hello');
      end;

-- call the file :
   @ .\test.sql
--or
   start .\test.sql

-- variables :
   identifier [constant] datatype [not null] [:= | default expr];
   -- WHEN you use no null you should give value :

--*example 1 :
   declare
   vDate date;
   vNo number:=10;
   vName varchar(100) not null:='ayoub';

--* example 2 :
   declare
   vDate date;
   vNo number:=10;
   vName varchar(100) not null:='ayoub';
   begin
   dbms_output.put_line('the date   : ' || vDate);
   dbms_output.put_line('the number :'  || vNo);
   dbms_output.put_line('the name   : ' || vName);

   vNo:=vNo + 10;
   dbms_output.put_line('the new  number :'  || vNo);

   vName:='majid';
   dbms_output.put_line('the new  name   : ' || vName);

   vDate:='10-feb-2023';
   dbms_output.put_line('the new  date   : ' || vDate);
   end;

-- get the current date :
   declare
```

```
   vCurrrentDate date :=sysdate;

--* example 1 :
   declare
   vDate date :=sysdate;
   vPi constant number:=3.14 ;
   begin
   vDate:= vDate + 10;
   dbms_output.put_line('the current date  date   : ' || vDate);
   dbms_output.put_line('the number :'  || vPi);
   end;

--  q notation (to espace special characters)
   q'('test')';
   --or
   q'['test']';
   --or
   q'x'test'x';
--*examaple 1:
   begin
   dbms_output.put_line(q'(father's day )');
   end;
--*examaple 2:
   begin
   dbms_output.put_line( q'[father's day ]' );
   end;
--*example 3:
   select q'[today is the father's day not the mother's day ]'
   from dual;

-- type of variables :
   /*
   Si une variable est déclarée avec l'option CONSTANTE, elle doit être initialisée
   Si une variable est déclarée avec l'option NOT NULL, elle doit être initialisée
   ✨NUMBER[(e,d)] Nombre réel avec e chiffres significatifs stockés et d décimales
   ✨PLS_INTEGER Nombre entier compris entre -2 147 483 647 et +2 147 483 647
   ✨CHAR [(n)] Chaîne de caractères de longueur fixe avec n compris entre 1 et 32767
   ✨VARCHAR2[(n)] Chaîne de caractères de longueur variable avec n compris entre 1 et 32767
   ✨BOOLEAN
   ✨DATE
   ✨RAW[[(n)] Chaîne de caractères ou données binaires de longueur variable
               avec n compris entre 1 et 32767. Le contenu
               'une variable de ce type n'est pas interprété par PL/SQL
   ✨LONG RAW Identique au type LONG qui peut contenir des données binaires
   ✨LONG Chaîne de caractères de longueur variable avec au maximum 32760 octets
   ✨ROWID Permet de stocker l'adresse absolue d'une ligne dans une table sous la
     forme d'une chaîne de caractères '
   */

-- interval year to month
--* example :
   DECLARE
   vName VARCHAR2(30) := 'ayoub';
   trip INTERVAL YEAR TO MONTH := INTERVAL '1-4' YEAR TO MONTH;
   BEGIN
   DBMS_OUTPUT.PUT_LINE('My age is: ' || EXTRACT(YEAR FROM age) || ' years ' ||
```

```
   EXTRACT(MONTH FROM age) || ' months');
   END;
   /

-- a type colonne:
   nom_variable nom_table.nom_colonne%TYPE ;

-- b reference from another variable
   nom_variable nom_variable_ref%TYPE ;

-- c type row :
   nom_variable nom_table%ROWTYPE;

--* example 1 :
   declare
   vTest test%rowtype ;
   begin
   select * into vTest  from test  fetch first  1 row only  ;
   dbms_output.put_line('the first name is : ' || vTest.firstName);
   dbms_output.put_line('the last name is : ' || vTest.lastName);
   dbms_output.put_line('the  the salary name is : ' || vTest.salary);
   end;
   /
--* example 2 :
   set serveroutput on;
   declare
      vflag  boolean;
      vno1   number default 21;
      vno2   number default 20;
      vprint varchar(100);
   begin
      vflag:=false;
      if vno1 =vno2 then
         vflag:=true;
         vprint:='numbers are equal';
      else
         vprint:='numbers are not equal';
      end if;
   -- print the stat
      dbms_output.put_line(vprint);
   end;
   /

-- if structre :
-- if
   IF condition THEN
   instruction1 ;
   instruction 2 ;
   ……..
   instruction 2 ;
   END IF;

-- if else
   IF condition1 THEN
   instruction1;
   instruction 2;
```

```
   ELSE
   instruction3;
   END IF;

-- if elsif  esle
   IF condition1 THEN
   instruction1;
   instruction 2;
   ELSIF condition2 THEN
   instruction 3;
   instruction 4;
   ELSIF condition3 THEN
   instruction 5;
   instruction 6;
   ELSE instruction 7;
   END IF;

-- relational operators :
   =
   <> or !=
   >
   <
   >=
   <=

-- logical operators :
   and
   or
   not
-- example 1:
   IF NOT (x = y) THEN
      -- Code to execute if x is not equal to y
   END IF;

-- bind variables delete
   variable  identifier [constant] datatype [not null] [:= | default expr];
   /*
   Bind variables are:
   . Created in the environment
   . Also called host variables
   . Created with the VARIABLE keyword
   . Used in SQL statements and PL/SQL blocks
   . Accessed even after the PL/SQL block is executed
   . Referenced with a preceding colon
   */
-- composite data type :
   TYPE nom_type_rec IS RECORD (
   nom_champ1 type_élément1 [[ NOT NULL] := expression ],
   nom_champ2 type_élément2 [[ NOT NULL] := expression ],
   …
   nom_champN type_élémentN[[ NOT NULL] := expression ]
   ) ;
   Nom_variable nom_type_rec;

-- put value of select query in a variable :
   select columm into variableName from tableName [where ...];
```

```
278
279 -- into  structure
280    select column1 ,column2 into s_Name from tableName;
281 -- or:
282    select column1 , column2 into s_Name.att1 ,s_Name.att2 from tableName;
283
284 -- Declare a bind variable named vSal
285    variable vSal NUMBER;
286
287 -- assign a default to a bind var (outside plsql block ) :
288    exec :bindVarName :=value;
289    -- or (throught pl sql block )
290    begin
291    :bindVarName :=value;
292    end;
293
294  -- print the value of  bind var :
295    print vSal
296
297 -- get the value of bind var tought the block
298    begin
299    dbms_output.put_line('the value of the  bind var  is : ' || :vName);
300    end;
301
302 --* example :
303    VARIABLE vSal NUMBER;
304    BEGIN
305    :vSal :=10;
306    dbms_output.put_line('hello ' || :vSal);
307    END;
308    /
309    @ test.sql;
310
311 -- print the value of bind var  after the block  automatically
312    set autoprint on;
313
314 --assignemnt 2 :
315    -- define a bind variale v_sal to be a number
316    -- create a bolck to store the salary for employee_id=1 in this variable
317    -- print the variable
318
319 -- PL/SQL block
320    BEGIN
321    SELECT salary INTO :vSal FROM test WHERE id = 1;
322    dbms_output.put_line('the salary is : ' || :vSal);
323    END;
324    /
325    print vSal;
326
327 -- wrting executable statements :
328    • Identifiers: v_empno, v_ename, " first Name "
329    • Delimiters : ; + -
330    • Literals: v ename='khaled'
331    • Comments: -- , /**/
332    -- simple symbols :
333    Symbol   -Meaning
```

```
   +       Addltron operator
   -       Subtraction/negation operator
   *       Multiplication operator
   /       Division operator
   =       Equality operator
   @       Remote access indicator
   ;       Statement temunator

-- Compound symbols
   Symbol    -Meaning
   <>      Inequality operator
   !=      Inequality operator
   ||      Concatenation operator
   '--'     Single-li comment indicator
   '/*'    Beginning comment delimiter
   */      Ending comment delimiter
   :=      Assignment operator

   Data Type Conversion
   • Converts data to comparable data types
   • Is of two types:
   Implicit conversion
   Explicit conversion
   • Functions:
   – TO_char
   – TO_date
   - TO_number
   – TO_timestamp

-- nested block :-----------------------------------
-- outer block
   declare
   BEGIN
      -- nseted block :
      declare
         begin
         dbms_output.put_line('hello world');
         end;
   END;
   /

--*example 1:
   declare
   vname varchar(30) :='ayoub';
   BEGIN
      declare

         begin
         dbms_output.put_line('hello world my name is : '||vname );
         end;
   END;
   /

--*example 2:
   /*
    access to global var from the nested block (in case when you have
```

```
    a local and global var with the same nane )
   */
   begin <<outer>>
   declare
   vname varchar(30) :='ayoub';
   BEGIN
      declare
   vname varchar(30) :='kyoub';
         begin
         dbms_output.put_line('hello world my name is : '||outer.vname );
         dbms_output.put_line('hello world my name is : '||vname );
         end;
   END;
   end outer;
   /
   select * from test;

-- SQL Statements in PL/SQL
   ✨ Retrieve a row from the database by using the SELECT
   command.
   ✨ Make changes to rows in the database by using DML
   commands.
   ✨ Control a transaction with the COMMIT, ROLLBACK, or
   SAVEPOINT command.
   ✨ PL/SQL does not directly support data definition language (DDL) statements,
   ✨ PL/SQL does not directly support data control language (DCL) statements,
   such as GRANT or REVOKE. You can use dynamic SQL to execute them.

--  SQL Cursor
   /*
   . A cursor is a pointer to the private memory area allocated by
   the Oracle server.
   . A cursor is used to handle the result set of a SELECT
   statement.
   · There are two types of cursors:
      ✨- Implicit: Created and managed internally by the Oracle
      server to process SQL statements
     ✨ - Explicit: Declared explicitly by the programmer

      SQL Cursor Attributes for Implicit Cursors

   Using SQL cursor attributes, you can test the outcome of your
   SQL statements.
   */
-- SQL%FOUND
  Boolean attribute that evaluates to TRUE if the
  most recent SQL statement returned at least one
  row

-- SQL%NOTFOUND
   Boolean attribute that evaluates to TRUE if
   the most recent SQL statement did not
   return even one row

-- SQL%ROWCOUNT
   --is only accurate after a statement that modifies data (e.g., INSERT, UPDATE, DELETE)
```

```
   An integer value that represents the number of
   rows affected by the most recent SQL statement

-- example rowcount :
   begin
   update test set salary =0 where id=1;
   dbms_output.put_line('the number of rows updated is : ' || sql%rowcount);
   end;

-- example found :
   declare
   vrowsExist boolean default false;
   begin
   update test set salary =0 where id=10;
   vrowsExist:= sql%found;
   if vrowsExist  then
   dbms_output.put_line('the stat  :  yes');
   else
   dbms_output.put_line('the stat  :  no');
   end if;
   end;
   /

-- change the prompt message ----------------------[-]:
   you can change the prompt as follow
   but it should executed as a script
   syntaxe :
   ACCEPT  Variable_name PROMPT 'Messgae' -- accept =define + prompt change message
   -- this message it will just assoscited with only this variable

--* example 1:
   ACCEPT User_id PROMPT 'Pelase enter the user id :';
   select employee_id,first_name,last_name,salary
   from employees
   where employee_id=&User_id;

--* example 2:
   select first_name,last_name,&&User_column
   from employees

   order by  &User_column;
--* example 3:
   ACCEPT User_column PROMPT 'Pelase enter the Column  :';
   select first_name,last_name,&User_column
   from employees
   order by  &User_column;


-- case expression :
   case selector
   when expression then result1
   when expression then result2
   ...
   when expression then resultN
   [else resultN+1]
   end;
```

```
   /
 --  case statements :
   case
   when condition1 then
   -- code
   when condition2 then
   -- code
   ...
   when conditionN then
   -- code
   [else
   -- code ]
   end case;
   /

--* example 1 :
   select firstname,lastname ,length(firstname) ,case length(firstname)
   when 4 then '4 char '
   when 5 then '5 char '
   when 6 then '6 char'
   else 'n/a'
   end
   from test;

--* example 2 :
   select firstname,lastname ,length(firstname) ,case
   when length(firstname) =4 then '4 char '
   when length(firstname)=5 then '5 char '
   when length(firstname) =6 then '6 char'
   else 'n/a'
   end
   from test;

--* example 3 :
   ACCEPT empId PROMPT 'Enter the employee id: ';
   DECLARE
      vSal NUMBER;
      vDesc VARCHAR(100);
   BEGIN
      SELECT salary INTO vSal FROM test WHERE id = &empId;

      vDesc := CASE
                  WHEN vSal IS NULL THEN 'No salary for the employee'
                  WHEN vSal BETWEEN 1000 AND 3000 THEN 'Salary is low'
                  WHEN vSal BETWEEN 3001 AND 5000 THEN 'Salary is medium'
                  WHEN vSal BETWEEN 5001 AND 10000 THEN 'Salary is good'
                  ELSE 'Salary is good'
               END;

      DBMS_OUTPUT.PUT_LINE('The employee status: ' || vDesc);
   END;
   /

--* example 3 : case statements
   ACCEPT empId PROMPT 'Enter the employee id: ';
   DECLARE
```

```
      vSal NUMBER;
      vDesc VARCHAR(100);
   BEGIN
      SELECT salary INTO vSal FROM test WHERE id = &empId;

   CASE
                  WHEN vSal IS NULL THEN
                     DBMS_OUTPUT.PUT_LINE('No salary for the employee');
                  WHEN vSal BETWEEN 1000 AND 3000 THEN
                     DBMS_OUTPUT.PUT_LINE('Salary is low');
                  WHEN vSal BETWEEN 3001 AND 5000 THEN
                     DBMS_OUTPUT.PUT_LINE('Salary is medium');
                  WHEN vSal BETWEEN 5001 AND 10000 THEN
                     DBMS_OUTPUT.PUT_LINE('Salary is good');
                  ELSE
                     DBMS_OUTPUT.PUT_LINE('Salary is good');
               END case;
   END;
   /

-- handling nulls value
   -- any comparaison  throught if  with null    :
   /*
   the if block :we not execut
   */

--  solution
   nvl(var,backValue)

--* example 1 :
   declare
   x number default 2;
   y number default null;
   begin
         if nvl(x,0) <> nvl(y,0) then
         dbms_output.put_line('hi');
      end if;
   end;
   /
-- · Loops repeat a statement (or sequence of statements)
      multiple times.
      <<label>>
      LOOP
         instruction1;
         instruction2;
         EXIT [label][WHEN condition1];
      END LOOP label;

      •EXIT force la sortie de la boucle sans conditions.
      •EXIT WHEN permet une sortie de boucle si la condition est vraie.
      •EXIT <<label>> WHEN permet une sortie d'une boucle nommée label si la condition
      est vraie.
      •EXIT <<label>> force une sortie de boucle nommée label.'

   --* example 1:  print value from 0 to 10:
      loop
```

```
            i:=i+1;
            dbms_output.put_line(i);
            exit when i=10;
         end loop;
      end;

   --* example 2 : multiplication table from 1 to 2 :
      set verify off;
      declare
         i integer:=0; j integer:=0;
      begin
      loop
      i:=i+1;j:=0;
         loop
            j:=j+1;
            dbms_output.put_line(i || '*' || j || '=' || i*j);
            exit when j=10;
         end loop;
         dbms_output.put_line(' ');
      exit when i=2;
      end loop;
      end;
      /

   --* example 3:  of exit in the first iteration : print multiplication talbe of 1
      set verify off;
      declare
         i integer:=0;
         j integer:=0;
      begin
      <<lable1>>
      loop
      i:=i+1;
      j:=0;
      <<lable2>>
         loop
            j:=j+1;
            dbms_output.put_line(i || '*' || j || '=' || i*j);
            exit lable1 when j=10;
         end loop lable2;
         dbms_output.put_line(' ');
      exit when i=2;
      end loop lable1;
      end;
      /

   --* example 4 : get the first name of employee 8 9 10:
      declare
      vfirstName  employees.first_name%type;
      vcounter integer  :=7;
      begin
      loop
         vcounter:= vcounter + 1;
      select  first_name into vfirstName from employees where employee_id=vcounter;
      dbms_output.put_line('the first name  of employee : ' || vfirstName);
      exit when vcounter=10;
```

```
      end loop;
      end;
      /

-- while loop:
      WHILE conditions
      LOOP
         instruction1;
         instruction2;
      END LOOP;
   --* example 1: print hello wolrd three times:
      declare
         vcounter integer :=1;
      begin
         while vcounter <= 3 loop
            dbms_output.put_line(vcounter || '-hello wolrd');
            vcounter:= vcounter +1;
         end loop;
      end;
      /

   --* example 2: print numbers from 0 to 10;
      while i<10 loop
         dbms_output.put_line(i);
         i:=i+1;
      end loop;

   --* example 3: -- get the first name of employee 8 9 10:
      declare
      vfirstName  employees.first_name%type;
      vcounter integer :=8;
      begin
      while vcounter <= 10 loop
         select first_name into vfirstName from employees
         where employee_id= vcounter;
         dbms_output.put_line(' the firstName  : ' || vfirstName);
         vcounter := vcounter +1;
      end loop;
      end;
      /

-- for in loop:
      FOR compteur IN [REVERSE] borne_inf..borne_sup LOOP
         instruction1 ;
         instruction2 ;
         instruction3 ;
      [EXIT WHEN condition];
      END LOOP;

   --* example 1 : print numbers from 1 to 5
      for  i in 1..5 loop
      dbms_output.put_line(i);
      end loop;

   --* example 2 : print numbers from 5 to 1 :
      for  i in reverse 1..5 loop
```

```
      dbms_output.put_line(i);
      end loop;


   --* example 3 : show just even numbers from 1 to 10 :
      DECLARE
      BEGIN
         FOR i IN 1..10 LOOP
            IF MOD(i, 2) = 1 THEN
                  -- Skip odd numbers
                  CONTINUE;
            END IF;

            DBMS_OUTPUT.PUT_LINE(i);
         END LOOP;
      END;
      /
   --* example 3 :
      FOR client_rec IN (SELECT NO, NOM, VILLE FROM E_CLIENT) LOOP
         DBMS_OUTPUT.PUT_LINE('Client Number: ' || client_rec.NO || ', Name: '
         || client_rec.NOM || ', City: ' || client_rec.VILLE);
      END LOOP;

   --* example 4 : print triangle :
      SET VERIFY OFF;
      ACCEPT inpn PROMPT 'Enter the size of triangle: ';
      DECLARE
         vn INTEGER := &inpn;
      BEGIN
         IF vn > 30 THEN
            DBMS_OUTPUT.PUT_LINE('The available size is 30');
         ELSE
            FOR i IN 1..vn LOOP
               FOR j IN 1..i LOOP
                  DBMS_OUTPUT.PUT('*');
               END LOOP;
               DBMS_OUTPUT.PUT_LINE('');
            END LOOP;
         END IF;
      END;
      /

-- write a program to print :
/*
    1
    :)
    2
    :)
    3
    :)
    4
    :)
    5
    :)
    6
    7
```

```
    8
    9
    10
*/
--* method 1 :
    dbms_output.put_line( chr(10) || 'first method : ');
        for i in 1..10 loop
    dbms_output.put_line(chr(9) || i);
            if i>=6 then
                continue;
                end if;
                dbms_output.put_line(chr(9) ||':)');
            end loop;

--* method 2 :
    dbms_output.put_line( chr(10) ||'second method : ');
        for i in 1..10 loop
            dbms_output.put_line(chr(9) || i);
            if i<=5 then
                dbms_output.put_line(chr(9) ||':)');
                end if;
            end loop;

-- ------------------------------------------------------------------------
Composite Data Types
    · Can hold multiple values (unlike scalar types)
    · Are of two types:
        - PL/SQL records
        - PL/SQL collections
            - INDEX BY tables or associative arrays
            - Nested table
            - VARRAY

Declaring a PL/SQL Record
    1- programmer-defined records.
    2- table-based record. %Rowtype
    3- cursor-based record. ( will be covered later )

--  1- programmer-defined records :
    TYPE nom_type_rec IS RECORD (
    nom_champ1 type_élément1 [[ NOT NULL] := expression ],
    nom_champ2 type_élément2 [[ NOT NULL] := expression ],
    …
    nom_champN type_élémentN[[ NOT NULL] := expression ]
    ) ;
--* example 1:
    declare
        type stemp is record (
            vempid employees.employee_id%type,
            vfirstname employees.first_name%type,
            vlastname employees.last_name%type
        );
        vemp stemp;
    begin
        select
            employee_id,
```

```
            first_name,
            last_name into vemp
        from
            employees
        where
            employee_id=1;
        dbms_output.put_line('the  id : ' || vemp.vempid);
        dbms_output.put_line('the first name : ' || vemp.vfirstname);
        dbms_output.put_line('the last  name : ' || vemp.vlastname);
    end;
/

-- create an empty copy from an existant table :
-- (notice : the copy table doesn't copy constraints excpet not null)
    create table copyTableName as select columns from mainTableName where 1=2;

    -- example 1: insert row to copyTable  using  a record
    create table copyEmp
    as select employee_id ,first_name,last_name from employees
    where 1=2;
    declare
        type stemp is record (
            vempid employees.employee_id%type,
            vfirstname employees.first_name%type,
            vlastname employees.last_name%type
        );
        vemp stemp;
    begin
        select employee_id, first_name, last_name into vemp from employees
        where employee_id=1;

    insert into copyEmp values vemp;
    end;
    /
    select * from copyEmp;


-- 2- table-based record. %Rowtype
   VarName tabName%rowtype;

   --* example 1: insert row to copyTable  using  a record
      create table copyEmp
      as select * from employees
      where 1=2;
      declare
      vemp employees%rowtype;
      begin
         select * into vemp from employees
         where employee_id=1;
      insert into copyEmp values vemp;
      end;
      /
      select * from copyEmp;


-- update row directly :
```

```
      update tableName;
      set row=stTabName;

   --*exmaple 1 :
      declare
         vemp employees%rowtype;
      begin
      vemp.employee_id :=10;
         vemp.first_name :='ayoub';
      update copyEmp
      set row=vemp;
      end;
      /
      select * from copyEmp;

--  PL/SQL collections ----------------------------------------------------

      - INDEX BY tables or associative arrays
      - Nested table
      - VARRAY

-- INDEX BY Tables or Associative Arrays
   /*
   . Are PL/SQL structures with two columns:
         - Primary key of integer or string data type
         - Column of scalar or record data type
   . Are unconstrained in size. However, the size depends on the
      values that the key data type can hold.

   */

   -- syntax :
      type tableName is table of  valueType
      index by [pls_integer | binary_integer | varchar2(size)];
   -- set  data :
      vtablename(key):=value;
   -- get data :
      vtableName(key);

   --example 1 :
      declare
      type arrtab is table of varchar2(100)
      index by pls_integer;
      vtab arrtab;
      begin
      vtab(-1):='KMoub';
      vtab(0):='KMoub';
      vtab(1):='ayoub';
      vtab(2):='amine';
      dbms_output.put_line(vtab(-1));
      dbms_output.put_line(vtab(0));
      dbms_output.put_line(vtab(1));
      dbms_output.put_line(vtab(2));
      end;
      /
-- table methods :
```

```
   vtablename.method([parameters]);
   --   EXISTS (n)
      .Returns TRUE if the nth element in a PL/SQL table exists

   -- COUNT
      .Returns the number of elements that a PL/SQL table currently

   -- FIRST
      · Returns the first (smallest) index number in a PL/SQL table
      . Returns NULL if the PL/SQL table is empty

   -- LAST
      · Returns the last (largest) index number in a PL/SQL table
      . Returns NULL if the PL/SQL table is empty

   -- PRIOR (n)
      .Returns the index number that precedes index n in a PL/SQL table

   -- NEXT (n)
      .Returns the index number that succeeds index n in a PL/SQL table

   -- DELETE
      ·DELETE removes all elements from a PL/SQL table.
      .DELETE (n) removes the nth element from a PL/SQL table.
      ·DELETE (m, n) removes all elements in the range m ... n from a pl/sql table

   --* example  1:
      declare
      type arrtab is table of varchar2(100)
      index by pls_integer;
      vtab arrtab;
      begin
         vtab(1):='KMoub';
         vtab(2):='KMoub';
         vtab(6):='KMoub';
         vtab(9):='ayoub';
         for i in 1..10 loop
            if vtab.exists(i) then
            dbms_output.put_line( chr(9)|| i || '-element exsit : ' || vtab(i));
            else
            dbms_output.put_line( chr(9)|| i ||'-element not  exsit :  -) ' );
            end if;
         end loop ;
         dbms_output.put_line('the total number of elements : ' || vtab.count());
         dbms_output.put_line('the first   element : ' || vtab.first());
         dbms_output.put_line('the next element index after index 2 : ' || vtab.next(2));
      end;
      /

   --* example 2 : rowtype
      declare
      type arrtab is table of employees%rowtype
      index by pls_integer;
      vtab arrtab;
      begin
         vtab(1).employee_id:=1;
```

```
1006          vtab(1).first_name:='ayoub';
1007          vtab(1).last_name:='majid';
1008          vtab(1).salary:=599;
1009
1010          dbms_output.put_line('the id : ' ||vtab(1).employee_id );
1011          dbms_output.put_line('the first name : ' ||vtab(1).first_name );
1012          dbms_output.put_line('the last name : ' ||vtab(1).last_name );
1013          dbms_output.put_line('the salary : ' ||vtab(1).salary );
1014       end;
1015       /
1016
1017    --* example 3 :
1018    declare
1019    type arrtab is table of employees%rowtype
1020    index by pls_integer;
1021    vtab arrtab;
1022    begin
1023       for i in  5..10 loop
1024          select * into vtab(i) from employees
1025          where employee_id=i;
1026       end loop;
1027       for i in vtab.first..vtab.last loop
1028          dbms_output.put_line( chr(9) ||'the id : ' ||vtab(i).employee_id );
1029          dbms_output.put_line( chr(9) ||'the first name : ' ||vtab(i).first_name );
1030          dbms_output.put_line( chr(9) ||'the last name : ' ||vtab(i).last_name );
1031          dbms_output.put_line( chr(9) ||'the salary : ' ||vtab(i).salary  || chr(10));
1032       end loop;
1033    end;
1034    /
1035
1036 -- nested tables :
1037   /*
1038    . No index in nested table ( unlike index by table )
1039    · It is valid data type in SQL ( unlike index by table, only used in PL/SQL )
1040    . Initialization required
1041    . Extend required
1042    · Can be stored in DB
1043    . start with index 1
1044   */
1045    --syntax
1046       type tableName is table of  valueType
1047
1048    --* example 1:
1049       declare
1050       type arrlocation is table of varchar2(100);
1051       loc arrlocation;
1052
1053       begin
1054       -- you should initialise it :
1055          loc:=arrlocation('Morocco','Jordan','UKA');
1056          dbms_output.put_line(loc(1));
1057          dbms_output.put_line(loc(2));
1058          dbms_output.put_line(loc(3));
1059       end;
1060    /
1061
```

```
    -- increase the size by one  :
    arrName.extend;
    -- example :
       declare
       type arrlocation is table of varchar2(100);
       loc arrlocation;

       begin

          loc:=arrlocation('Morocco','Jordan','UKA');
          loc.extend;
          loc(4):='uk';
          dbms_output.put_line(loc(1));
          dbms_output.put_line(loc(2));
          dbms_output.put_line(loc(3));
          dbms_output.put_line(loc(4));
       end;
       /

    -- delete  element :
       arrName.delete(index);
    --example :
    declare
    type arrlocation is table of varchar2(100);
    loc arrlocation;

    begin
       loc:=arrlocation('Morocco','Jordan','UKA');
       loc.extend;
    loc.delete(1);
       dbms_output.put_line(loc(2));
       dbms_output.put_line(loc(3));
    end;
    /

-- VARRAY : like nested array but with fixed size
    --syntax
       type tableName is varray(size) of  valueType

    --*example 1:
          declare
          type arrlocation is varray(3) of varchar2(100);
          loc arrlocation;

          begin
          -- you should initialise it :
             loc:=arrlocation('Morocco','Jordan','UKA');

             dbms_output.put_line(loc(1));
             dbms_output.put_line(loc(2));
             dbms_output.put_line(loc(3));
          end;
       /


    -- pop element :
```

```
      arrName.trim(nbrOfElemels);
   --*examaple :
      declare
            type arrlocation is varray(3) of varchar2(100);
            loc arrlocation;

            begin
            -- you should initialise it :
               loc:=arrlocation('Morocco','Jordan','UKA');
               loc.trim(2);

               dbms_output.put_line(loc(1));

            end;
         /


-- using Explicit cursor ------------------------------------------------------
   /*
   Every SQL statement executed by the Oracle server has an
   associated individual cursor:
   . Implicit cursors: Declared and managed by PL/SQL for all
   DML and PL/SQL SELECT statements
   . Explicit cursors: Declared and managed by the programmer
   */
-- Explicit Cursor Operations
   /*
      You declare explicit cursors in PL/SQL when you have a SELECT statement that returns
multiple
      rows. You can process each row returned by the SELECT statement.
   */
--Explicit cursor functions:
   /*
   . Can perform row-by-row processing beyond the first row returned by a query
   . Keep track of the row that is currently being processed
   . Enable the programmer to manually control explicit cursors in the PL/SQL block
   */

--define a cursor :
   cursor cursorName is (select columns from tableName [where ...]);

-- open a cursor :
   open cursorName;
 /*
   The OPEN statement executes the query associated with the cursor, identifies the active
set, and
   positions the cursor pointer at the first row. The OPEN statement is included in the
executable section
   of the PL/SQL block.
   OPEN is an executable statement that performs the following operations:
   1. Dynamically allocates memory for a context area
   2. Parses the SELECT statement
   3. Binds the input variables (sets the values for the input variables by obtaining their
memory
   addresses)
   4. Identifies the active set (the set of rows that satisfy the search criteria). Rows in
the active
```

```
    set are not retrieved into variables when the OPEN statement is executed. Rather, the FETCH statement
    retrieves the rows from the cursor to the variables.
    5. Positions the pointer to the first row in the active set
    Note: If a query returns no rows when the cursor is opened, PL/SQL does not raise an exception.
    You can find out the number of rows returned with an explicit cursor by using the
    <cursor name>%ROWCOUNT attribute.
 */

-- Fetching Data from the Cursor
 fetch cursorName into vDatName;
 /*
    The FETCH statement retrieves the rows from the cursor one at a time. After each fetch, the cursor
    advances to the next row in the active set. You can use the %NOTFOUND attribute to determine
    whether the entire active set has been retrieved.

    The FETCH statement performs the following operations:
    1. Reads the data for the current row into the output PL/SQL variables
    2. Advances the pointer to the next row in the active set
 */

--  Closing the Cursor
close cursorName;
  /*
    The CLOSE statement disables the cursor, releases the context area, and "undefines" the active set.
    Close the cursor after completing the processing of the FETCH statement. You can reopen the cursor
    if required. A cursor can be reopened only if it is closed. If you attempt to fetch data from a cursor
    after it has been closed, then an INVALID_CURSOR exception will be raised.
    Note: Although it is possible to terminate the PL/SQL block without closing cursors, you should
    make it a habit to close any cursor that you declare explicitly to free up resources.
    There is a maximum limit on the number of open cursors per session, which is determined by the
    OPEN_CURSORS parameter in the database parameter file. (OPEN_CURSORS = 50 by default.)
  */

--cursorname%ISOPEN 1
    Evaluates to TRUE if the cursor is open

-- cursorname%FOUND
  Boolean attribute that evaluates to TRUE if the
  most recent SQL statement returned at least one
  row

-- cursorname%NOTFOUND
    Boolean attribute that evaluates to TRUE if
    the most recent SQL statement did not
    return even one row

-- cursorname%ROWCOUNT
  --is only accurate after a statement that modifies data (e.g., INSERT, UPDATE, DELETE)
    An integer value that represents the number of
```

```
   rows affected by the most recent SQL statement

--* example 1 :
   set serveroutput on;
   declare
      cursor cEmp  is
      select * from employees where employee_id between 1 and 6;
      vEmp  employees%rowtype;
   begin
      -- opent the cursor :
      open cEmp;

   dbms_output.put_line(chr(10)|| ' ----employees dat from 1 to 6---- ' || chr(10));
         -- fetch the first row from the cursor :
         fetch cEmp into vEmp;

         while cEmp%found loop
               -- print the data of employee :
               dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
               dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
               dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
               dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary || chr(10));

               -- fetch again till the end :
               fetch cEmp into vEmp;
         end loop;
      -- close the cursor :
      close cEmp;
   end;
   /

--  define a variable of cursor :
   declare
   varNamee cursorname%rowtype;

--* example 2 : using for loop
   /*
   1-cursor open automatically
   3- auto fetch
   2-cursor close automatically
   */
   set serveroutput on;
   declare
      cursor cEmp  is
      select * from employees where employee_id between 1 and 6;
   begin

   dbms_output.put_line(chr(10)|| ' ---- employees dat from 1 to 6---- ' || chr(10));

         for  vEmp in cEmp  loop
               -- print the data of employee :
               dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
               dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
               dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
               dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary || chr(10));
         end loop;
```

```
   end;
   /

--* exmaple 3 :  increase the salary by 100$ for employees where  the id between  1 and 6
   select * from employees
   where employee_id  between 1 and 6;
   set serveroutput on;
   declare
      cursor cEmp  is
      select * from employees where employee_id between 1 and 6;
   begin
   dbms_output.put_line(chr(10)|| ' ---- update employees dat from 1 to 6---- ' || chr(10));

      for  vEmp in cEmp  loop
         update employees
         set salary=salary +100
         where  employee_id=vEmp.employee_id;
      end loop;
      commit;
   end;
   /
   select * from employees
   where employee_id  between 1 and 6;

--*example 4 :
   set serveroutput on;
   declare
      cursor cEmp  is
      select * from employees;
      vEmp cEmp%rowtype;
   begin
      if cEmp%isopen then
         null;
      else
         open cEmp;
      end if;
      dbms_output.put_line('the counter for cursor now is : ' || cEmp%rowcount);
      loop
         fetch cEmp into vEmp;
         exit when cEmp%notfound or cEmp%rowcount >6 ;
         dbms_output.put_line('the counter for cursor now is : ' || cEmp%rowcount);
      end loop;
   end;
   /

-- cursor with parameters :
   cursor cursorName(arg1 datatype ...) is (select columns from tableName [where ...]);
 /*

   Parameter data types are the same as those for scalar variables, but you do not give them
sizes.

   You can pass parameters to the cursor that is used in a cursor FOR loop:
   DECLARE
   CURSOR c_emp_cursor (p_deptno NUMBER, p_job VARCHAR2) IS
   SELECT
```

```
   BEGIN
   FOR emp_record IN c_emp_cursor (10, 'Sales') LOOP ...
 */
...

--* example  1: loop
   set serveroutput on;
   declare
      cursor cEmp(vDep number)  is
      select * from employees where dep=vdep;
      vEmp cEmp%rowtype;
   begin
         open cEmp(1);
            dbms_output.put_line(chr(10)|| ' ---- dep 1 --- ' || chr(10));
      loop
         fetch cEmp into vEmp;
         exit when cEmp%notfound or cEmp%rowcount >6 ;
            dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
         dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
         dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
         dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary );
         dbms_output.put_line(chr(9) ||'the departemment  : ' || vEmp.dep || chr(10));
      end loop;
      close cEmp;

         open cEmp(2);
         dbms_output.put_line(chr(10)|| ' ---- dep 2 --- ' || chr(10));
      loop
         fetch cEmp into vEmp;
         exit when cEmp%notfound or cEmp%rowcount >6 ;
            dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
         dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
         dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
         dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary );
         dbms_output.put_line(chr(9) ||'the departemment  : ' || vEmp.dep || chr(10));
      end loop;
      close cEmp;

         open cEmp(3);
         dbms_output.put_line(chr(10)|| ' ---- dep 3 --- ' || chr(10));
      loop
         fetch cEmp into vEmp;
         exit when cEmp%notfound or cEmp%rowcount >6 ;
            dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
         dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
         dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
         dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary );
         dbms_output.put_line(chr(9) ||'the departemment  : ' || vEmp.dep || chr(10));
      end loop;
      close cEmp;

   end;
   /

--*example 2 : for loop
   set serveroutput on;
```

```
   declare
      cursor cEmp(vDep number)  is
      select * from employees where dep=vdep;
      vEmp cEmp%rowtype;
   begin
            dbms_output.put_line(chr(10)|| ' ---- dep 1 --- ' || chr(10));
      for vEmp in cEmp(1)
      loop
            dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
         dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
         dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
         dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary );
         dbms_output.put_line(chr(9) ||'the departemment  : ' || vEmp.dep || chr(10));
      end loop;


         dbms_output.put_line(chr(10)|| ' ---- dep 2 --- ' || chr(10));
      for vEmp in cEmp(2)
      loop
            dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
         dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
         dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
         dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary );
         dbms_output.put_line(chr(9) ||'the departemment  : ' || vEmp.dep || chr(10));
      end loop;

         dbms_output.put_line(chr(10)|| ' ---- dep 3 --- ' || chr(10));
         for vEmp in cEmp(3)
      loop
            dbms_output.put_line(chr(9) ||'the id  : ' || vEmp.employee_id);
         dbms_output.put_line(chr(9) ||'the first name : ' || vEmp.first_name);
         dbms_output.put_line(chr(9) ||'the last name  : ' || vEmp.last_name);
         dbms_output.put_line(chr(9) ||'the salary  : ' || vEmp.salary );
         dbms_output.put_line(chr(9) ||'the departemment  : ' || vEmp.dep || chr(10));
      end loop;

   end;
   /


-- for update clause :
    cursor cursorName is (select columns from tableName [where ...])
    for update [of column1,column2... | nowait | wait duration];
  /*
   .nowait :  don't wait just return an error
   .wait 30 :  return error after 30 second  else execute
   . Use explicit locking to deny access to other sessions for the
   duration of a transaction.(unti commit or rollback)
   . Lock the rows before the update or delete.
  */

-- use it rather than  : where table_id=record.id;
    where current of cursorName;

 --*example 1 :
      select * from employees where dep in (2,3);
```

```
   set serveroutput on;
   declare
      cursor cEmp  is
      select * from employees where dep in (2,3) for update  of salary;
      vEmp cEmp%rowtype;
   begin

         dbms_output.put_line(chr(10)|| ' ---- dep 1 & 2 --- ' || chr(10));
         for vEmp in cEmp
         loop
         update employees
         set salary= salary -100
         -- use it rather than  : where employee_id=vEmp.employee_id;
         where current of cEmp;
         end loop;
      commit;
   end;
   /
    select * from employees where dep in (2,3);



-- handling exception -----------------------------------
 /*
   . An exception is a PL/SQL error that is raised during program
      execution.
   . An exception can be raised:
   - Implicitly by the Oracle server
   - Explicitly by the program
   . An exception can be handled:
      - By trapping it with a handler
      - By propagating it to the calling environment
 */

--  syntax :
   EXCEPTION
 WHEN exception1 [OR exception2 . . .] THEN
      statement1;
      statement2;

   [WHEN exception3 [OR exception4 . . .] THEN
      statement1;
      statement2;
   .]
   [WHEN OTHERS THEN
      statement1;
      statement2;
   . . .J

/*
   . The EXCEPTION keyword starts the exception-handling
   section.
   . Several exception handlers are allowed.
   · Only one handler is processed before leaving the block.
   . WHEN OTHERS is the last clause.

```

```
*/
--most common defined exceptions
  /*
   . Reference the predefined name in the exception-handling
   routine.
   · Sample predefined exceptions:
   - NO_DATA_FOUND
   - TOO_MANY_ROWS
   - INVALID_CURSOR
   - ZERO_DIVIDE
   - DUP_VAL_ON_INDEX (insert with duplicate primary key)
  */

--* example 1 :
   declare
   vLastname varchar(30);
   begin
      select last_name into vLastname from employees
      where employee_id=12/0;
      dbms_output.put_line('the last name : ' ||vLastname);

      exception
         when no_data_found then
            dbms_output.put_line('the query doesn''t retrieve any record ');
         when too_many_rows then
            dbms_output.put_line('the query retrieve more than one record ');
         when zero_divide then
            dbms_output.put_line('divide by 0 not allowed');
         when others then
         dbms_output.put_line('other Error');
   end;
   /

--* example 2 :
   declare
   vfirstName employees.first_name%type;
   begin
      for i in  5..12  loop
         begin
               select first_name into vfirstName from employees
               where employee_id=i;
               dbms_output.put_line('the first name : ' || vfirstname);

               exception
               when no_data_found then
               null;
         end;
      end loop;
   exception
         when no_data_found then
         null;
   end;
   /

-- declare  exception :
   vexpcetionName exception ;
```

```
1554
1555 -- add code to expcetion :
1556    prgram exception_init(vexceptionName,code);
1557
1558 -- show excpetion info :
1559   exception
1560   when exception then
1561   dbms_output.put_line(sqlerrm); -- message
1562   dbms_output.put_line(sqlcode);-- code default 1
1563
1564 -- example 1 :
1565    declare
1566       expInsertNull exception;
1567       pragma exception_init(expInsertNull,-1400);
1568    begin
1569
1570       insert into employees values
1571       (null,'','','10-jan-23',0,0);
1572
1573       exception
1574       when expInsertNull then
1575          dbms_output.put_line('Error ');
1576          dbms_output.put_line(sqlerrm);
1577          dbms_output.put_line('the code : ' || sqlcode);
1578       when others then
1579             null;
1580    end;
1581    /
1582
1583 --* example 2 :
1584    declare
1585       expInsertNull exception;
1586       expInvalidNumber exception;
1587       pragma exception_init(expInsertNull,-1400);
1588       pragma exception_init(expInvalidNumber ,-1722);
1589    begin
1590       begin
1591          insert into employees values
1592          (null,'','','10-jan-23',0,0);
1593          exception
1594          when expInsertNull then
1595                dbms_output.put_line('Error ');
1596                dbms_output.put_line(sqlerrm);
1597                dbms_output.put_line('the code : ' || sqlcode);
1598       end;
1599
1600       begin
1601          update employees
1602          set employee_id='ss'
1603          where employee_id=1;
1604          exception
1605          when expInvalidNumber  then
1606                dbms_output.put_line('Error ');
1607                dbms_output.put_line(sqlerrm);
1608                dbms_output.put_line('the code : ' || sqlcode);
1609       end;
```

```
1610
1611        exception
1612        when expInsertNull then
1613           dbms_output.put_line('Error ');
1614           dbms_output.put_line(sqlerrm);
1615           dbms_output.put_line('the code : ' || sqlcode);
1616        when others then
1617              null;
1618     end;
1619     /
1620  -- user defined excpetion :-----------------------------------------------------------------
     ----
1621  -- raise exception :
1622        raise vexceptionName;
1623
1624  -- raise exception without define exception section :
1625   /*
1626     Is a user-specified number for the exception between -20.000
1627     and -20.999
1628     Is the user-specified message for the exception; is a character string
1629     up to 2.048 bytes long
1630
1631     Is an optional Boolean parameter (If TRUE, the error is placed
1632     on the stack of previous errors. If FALSE, which is the default, the
1633     error replaces all previous errors.)
1634   */
1635    raise_application_error(errorCode,'msg');
1636
1637  --* example 1:
1638     declare
1639        vempId number  := 11;
1640        expinvalId exception ;
1641     begin
1642        update employees
1643        set salary =2000
1644        where employee_id =vempId;
1645
1646        if sql%notfound then
1647           raise expinvalId;
1648        end if;
1649
1650        exception
1651        when expinvalId then
1652           dbms_output.put_line('id not found ');
1653     end;
1654     /
1655
1656  --* example 2 :
1657     declare
1658        vempId number  := 11;
1659     begin
1660        update employees
1661        set salary =2000
1662        where employee_id =vempId;
1663
1664        if sql%notfound then
```

```
         raise_application_error(-20000,'id not found ');
      end if;
   end;
   /

-- procedure : ----------------------------------------
-- syntax :
   create or replace procedure procedureName
   [(par1 in datatype ... )] --parameters without size :
   is
   --declare variables
   begin
   [excpetion]

   end;

-- call a procedure :
   --outside the plsql block (begin end ) :
      execute procedureName(parametersValues);
   -- in plsql block :
      procedureName(parametersValues);

--* example 1 : create a procedure that print hello world
  -- create hello procedure :
   create or replace procedure printHello
   is
      begin
         dbms_output.put_line('hello world ');
      end;


   begin
   -- call the procedure :
      printHello();
   end;
   /

--* example 2 : udpate the salary of an employee
   create or replace procedure updateEmpSalary
   (id in number , eAmount in number)
   is
   expNegativeSalary exception;
      begin

      if eAmount < 0 then
         raise expNegativeSalary;
      end if;

      -- if it's ok update the salary :
         update employees
         set salary = eAmount
         where employee_id=id;
               exception
               when no_data_found then
               dbms_output.put_line('invalid id Try again ');
               when expNegativeSalary then
```

```
1721                dbms_output.put_line('You should enter a positive amount');
1722                when others then
1723                dbms_output.put_line('the code ' || sqlcode);
1724                dbms_output.put_line('the Message  ' || sqlerrm);
1725       end;
1726
1727    select * from employees where employee_id=1;
1728    begin
1729       updateEmpSalary(1,-500);
1730       dbms_output.put_line('the salary after  udpated : ');
1731    end;
1732    /
1733
1734    select * from employees where employee_id=1;
1735
1736 --* example 2 : using execute :
1737    select * from employees where employee_id=1;
1738    execute updateEmpSalary(1,-500);
1739    select * from employees where employee_id=1;
1740
1741 --* example 2 : reading inputs from user
1742    set verify off;
1743    accept inpEmpId prompt 'ente the employee id : '
1744    accept inpAmount prompt 'ente the  new amount  : '
1745    variable vEmpId number;
1746    exec :vEmpId :=&inpEmpId;
1747
1748    select * from employees where employee_id=1;
1749
1750    declare
1751       vAmount number  :=&inpAmount;
1752    begin
1753       updateEmpSalary(:vEmpId,vAmount);
1754       dbms_output.put_line('the salary after  udpated : ');
1755    end;
1756    /
1757    select * from employees where employee_id=1;
1758
1759 -- find a procedure  :
1760    select * from user_objects
1761    where object_name ='PROCEDURENAME';
1762
1763 -- find source code of a procedure :
1764    select * from user_source
1765    where name ='PROCEDURENAME';
1766
1767 -- drop procedure :
1768    drop procedure procedureName;
1769
1770 -- parameter-passing mode:
1771  /*
1772    - An IN parameter mode (the default) provides values for a
1773    subprogram to process
1774    - An OUT parameter mode returns a value to the caller pass (pass  by  ref and clear
   passed value )
1775    - An IN OUT parameter mode supplies an input value,
```

```
   which may be returned (output) as a modified value  (preserve the initial value)

 */

 --* example 1 : out  get the firstname and lastname of an employee :
     create or replace procedure  getEmpFullname
    (vEmpId number ,vFirstname  out employees.first_name%type, vLastname out
employees.last_name%type)
    is
    begin

        select first_name,last_name into vFirstname,vLastname from employees
        where employee_id=vEmpId;
        exception
            when no_data_found then
            dbms_output.put_line( chr(10) || 'invalid id [' ||vEmpId || '] try again' ||
chr(10));
            when others then
            dbms_output.put_line('');
            dbms_output.put_line('the code ' || sqlcode);
            dbms_output.put_line(' tje msg ' || sqlerrm);
            dbms_output.put_line('');
    end;

   declare
      vFirstname employees.first_name%type;
      vLastname employees.last_name%type;
   begin
      getEmpFullname(2,vFirstname,vLastname);

      dbms_output.put_line('the first name is : ' || vFirstname);
      dbms_output.put_line('the first name is : ' || vLastname);
   end;
   /

--* example 2 : in out  :  format phone
   create or replace procedure formatNumber
   (phone in out varchar)
   is
   begin
      phone := '( ' || phone ||' )';
   end;

   declare
      vphone varchar(12) :='07715633';
   begin
      dbms_output.put_line('the phone before format : ' || vphone);
      formatNumber(vphone);
      dbms_output.put_line('the phone after format : ' || vphone);

   end;
   /

-- default value :  just for in variables
-- syntax :
   create or replace procedure procedureName
   [(par1 in datatype:=defaultValue | default defaultValue  )] --parameters without size :
```

```
   is
   --declare variables
   begin
   [excpetion]

   end;

-- Available Notations forPassing Actual Parameters
 /*
   When calling a subprogram, you can write the actual
   parameters using the following notations:
   · Positional:
      - Lists the actual parameters in the same order as the
      formal parameters
   . Named:
      - Lists the actual parameters in arbitrary order and uses
        the association operator (=>) to associate a named formal
        parameter with its actual parameter
   . Mixed:
      - Lists some of the actual parameters as positional and
      some as named
 */

 --* example :
   create table products
   (
   prod_id number,
   prod_name varchar(30),
   prod_type varchar(20),
   constraint product_pk primary key(prod_id)
   );

   create or replace procedure add_prodact
   (vpro_id number , vpro_name varchar2 , vpro_type varchar2 :='sw' )
      is
      begin
         insert into products values
         (vpro_id,vpro_name,vpro_type);
         commit;

         exception
         when others then
         dbms_output.put_line('error in insert ');
         dbms_output.put_line(sqlcode);
         dbms_output.put_line(sqlerrm);

      end;
   --Positional
      execute add_prodact(2,'laptop');
   --Named
      execute add_prodact(vpro_id=>3, vpro_name=>'laptop',  vpro_type=>'tech');
      execute add_prodact( vpro_name=>'laptop', vpro_id=>4 ,vpro_type=>'tech');
   --  Mixed
      execute add_prodact(5,vpro_name=>'laptop',vpro_type=>'tech');

   select * from products;
```

```
1887
1888 -- functions ---------------------------
1889  /*
1890    . Is a named PL/SQL block that returns a value
1891    . Can be stored in the database as a schema object for
1892       repeated execution
1893    . Is called as part of an expression or is used to provide
1894       a parameter value
1895  */
1896
1897  --syntax :
1898  -- syntax :
1899    create or replace function functionName
1900    [(par1 in datatype ... )] --parameters without size :
1901     return datatype
1902    is
1903    --declare variables
1904    begin
1905    -- code
1906    return expression;
1907    [excpetion]
1908
1909    end;
1910
1911 --* example 1 :
1912    CREATE OR REPLACE FUNCTION get_sal(vemp_id NUMBER)
1913       RETURN NUMBER
1914       AS
1915       vSal NUMBER;
1916       BEGIN
1917       SELECT salary INTO vSal
1918       FROM employees
1919       WHERE employee_id = vemp_id;
1920
1921       RETURN vSal;
1922
1923       EXCEPTION
1924       WHEN NO_DATA_FOUND THEN
1925          DBMS_OUTPUT.PUT_LINE('No employee found with ID [' || vemp_id || ']');
1926          RETURN -1; -- Return NULL instead of 0 when no data is found
1927
1928       WHEN OTHERS THEN
1929          DBMS_OUTPUT.PUT_LINE('Error code: ' || SQLCODE);
1930          DBMS_OUTPUT.PUT_LINE('Error message: ' || SQLERRM);
1931          RETURN -1; -- Return NULL for other exceptions
1932       END;
1933       /
1934
1935
1936    declare
1937    vSal number ;
1938    begin
1939       vSal :=get_sal(12);
1940       dbms_output.put_line('the salary is : ' || vSal);
1941    end;
1942    /
```

```
   select get_sal(99) from dual;

--  you you want  to use function with sql :
 /*
  the function should not be contains any DML (select,insert,update,delete) , commit
rollback
 */
 --* example 2 :
   CREATE OR REPLACE FUNCTION get_Tax(p_sal NUMBER)
      RETURN NUMBER
      AS

   BEGIN
      if p_sal <5000 then
         return p_sal *(10/100);
         else
         return p_sal *(15/100);
      end if;
      end;

   begin
      dbms_output.put_line('the tax  :' || get_Tax(5000));
   end;
   /
   select employee_id,first_name ,get_Tax(salary)
   from employees;

   select * from employees
   where get_Tax(salary) > 1000;

-- sequence :
   /*
   In database systems, a sequence is an object that generates a series of unique values.
   Sequences are often used to generate primary key values for tables. Here are the key
   aspects of sequences
   */


-- create a sequence :
    CREATE SEQUENCE sequence_name [INCREMENT BY n | NOMAXVALUE] [START WITH n]
    [MAXVALUE n | NOMAXVALUE];

-- Retrieving Sequence Values:
   /*
   The NEXTVAL and CURRVAL functions are used to retrieve the next value
   and the current value
   of a sequence, respectively.
   */
-- Example 1:
    SELECT sequence_name.NEXTVAL FROM DUAL;

-- reset  a sequence :
    ALTER SEQUENCE sequence_name RESTART;

-- or (into the plsql program )
    EXECUTE IMMEDIATE 'ALTER SEQUENCE inc RESTART';
```

```
1998
1999 -- drop a sequence :
2000     DROP SEQUENCE sequence_name;
2001
2002
2003 --* example 1:
2004     CREATE TABLE example_table (
2005     id NUMBER PRIMARY KEY,
2006     data VARCHAR2(50)
2007     );
2008
2009     INSERT INTO example_table VALUES (sequence_name.NEXTVAL, 'Some data');
2010
2011 --* example 2:
2012     -- Create sequence with START WITH 1
2013     CREATE SEQUENCE inc START WITH 1;
2014
2015     -- Use the sequence to get the current value
2016     SELECT inc.currval FROM dual;
2017
2018     -- Use the sequence to get the next value
2019     SELECT inc.NEXTVAL FROM dual;
2020
2021     -- Declare a variable for first name
2022     DECLARE
2023     v_firstname usert.firstname%TYPE;
2024
2025     BEGIN
2026     -- Restart the sequence
2027     EXECUTE IMMEDIATE 'ALTER SEQUENCE inc RESTART';
2028     dbms_output.put_line(inc.NEXTVAL);
2029
2030     -- Select the first name into the variable
2031     SELECT firstname INTO v_firstname FROM usert WHERE userid = 1;
2032
2033     -- Display the first name
2034     dbms_output.put_line('The first name is: ' || v_firstname);
2035     END;
2036     /
2037
2038
2039 -- packages : ---------------------------------------------
2040 -- package specification :
2041    create or replace package packageName
2042    is
2043    function functionName(par1 datatype)
2044    return datatype;
2045    function functionNam2(par1 datatype)
2046    return datatype;
2047    ...
2048    [begin] -- begin run first when you create a package
2049    end;
2050
2051 -- package  body :
2052    create or replace package body packageName
2053    is
```

```
   function functionName(par1 datatype)
      return datatype
      is
      begin
      -- code
      end;
   function functionNam2(par1 datatype)
      return datatype
      is
      begin
      -- code
      end;
      ...

   end;

-- call a function from a package :
   packageName.functionName(arguements);

--* example 1 :
   create or replace package aria
   is
      function  retangleAria
         ( width number , height number)
         return number;

      function  squareAria( side number)
         return number;
   end;

   create or replace package body aria
   is
      function  retangleAria
      ( width number , height number)
      return number
      as
         begin
         return width*height;
         exception
               when invalid_number  then
                  dbms_output.put_line('you should enter a number ');
                  return null;
               when others then
                  raise_application_error(sqlcode,sqlerrm);
                  return null;
         end;
      function  squareAria( side number)
      return number
         as
         begin
         return side*2;
         exception
               when invalid_number  then
                  dbms_output.put_line('you should enter a number ');
                  return null;
               when others then
```

```
                  raise_application_error(sqlcode,sqlerrm);
                  return null;
         end;
   end;



select aria.retangleAria(5,4) from dual;

--* example 3 : package with variables
   create or replace package body aria
   is
      function  retangleAria
      ( width number , height number)
      return number
      as
         begin
         return width*height;
         exception
               when invalid_number  then
                  dbms_output.put_line('you should enter a number ');
                  return null;
               when others then
                  raise_application_error(sqlcode,sqlerrm);
                  return null;
         end;
      function  squareAria( side number)
      return number
         as
         begin
         return side*2;
         exception
               when invalid_number  then
                  dbms_output.put_line('you should enter a number ');
                  return null;
               when others then
                  raise_application_error(sqlcode,sqlerrm);
                  return null;
         end;
   end;

   create or replace function  get_mile_to_km
   (p_value number)
   return number
   is
      begin
      return global_Measurement.c_mile_to_km *  p_value ;
      end;
      /
   create or replace function  get_km_to_mille
   (p_value number)
   return number
   is
      begin
         return global_Measurement.c_km_to_mile *  p_value ;
      end;
```

```
      /

   create or replace package global_Measurement
   is
   c_mile_to_km constant number :=1.6093;
   c_km_to_mile constant number :=0.6214;
   end;

   execute dbms_output.put_line('60 mile =' ||  get_mile_to_km (60) || 'KM');
   execute dbms_output.put_line('100 KM =' || get_km_to_mille(100) || 'Mille');

--* example 4 : declare a function in a subprogram :
   declare
      function get_sysdate
      return date
      is
      begin
      return sysdate ;
      end;
   begin
      dbms_output.put_line(get_sysdate);
   end;
   /

-- drop a package :
  --drop specification :
   drop package packageName;
   --drop body :
   drop package body packageName;

-- stop 53 skip until ... 86

-- triggers : ---------------------------------------------:
   /*
      . A trigger is a PL/SQL block that is stored in the
      database and fired (executed) in response to a
         specified event.
      . The Oracle database automatically executes a trigger
      when specified conditions occur.
   */
   /*
      You can write triggers that fire whenever one of the
      following operations occurs in the database:
      · A database manipulation (DML) statement (DELETE,
      INSERT, or UPDATE).
      . A database definition (DDL) statement (CREATE, ALTER,
      or DROP).
      · A database operation such as SERVERERROR, LOGON,
      LOGOFF, STARTUP, or SHUTDOWN.
   */

--syntax :
   create or replace trigger triggerName
   before | after | instead of |
   insert | update | delete  -- you combine using or
   on tableName
```

```
   begin
   --  code
   if inseting then -- check if the event is insert
   if deleting then -- check if the event is delete
   if updating then -- check if the event is update
   -- code
   end;

-- convert date to heour :
   to_number(to_char(date,'hh24'));

-- extract HH:mm::ss
   SELECT
   TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) AS current_hour,
   TO_NUMBER(TO_CHAR(SYSDATE, 'MI')) AS current_minute,
   TO_NUMBER(TO_CHAR(SYSDATE, 'SS')) AS current_second
   FROM DUAL;

-- extract YYYY::MM::dd
   SELECT
   TO_CHAR(SYSDATE, 'YYYY') AS current_year,
   TO_CHAR(SYSDATE, 'MM') AS current_month,
   SELECT TO_CHAR(SYSDATE, 'DD') AS current_day FROM DUAL;
   FROM DUAL;

-- now day name :
   select TO_CHAR(SYSDATE, 'Day') AS day_name
   from dual;

   --* example 1 :
      CREATE OR REPLACE TRIGGER testTrig_check_time
   BEFORE INSERT OR UPDATE OR DELETE
   ON test
   BEGIN
      IF TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) NOT BETWEEN 8 AND 13 or
TO_NUMBER(TO_CHAR(SYSDATE, 'mi'))>=30
      THEN
         RAISE_APPLICATION_ERROR(-20010, 'DML Operations not allowed now');
      END IF;
   END;
   /

   BEGIN
      INSERT INTO test VALUES (2, 'amine', 'km', 500);
   END;
   /

--* example 2 : with  condition predicates :
-- Create the trigger
   CREATE OR REPLACE TRIGGER testTrig_check_time
   BEFORE INSERT OR UPDATE OR DELETE
   ON test
   BEGIN
      IF TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) NOT BETWEEN 8 AND 13 or
TO_NUMBER(TO_CHAR(SYSDATE, 'mi'))>=30
      THEN

```

```
         if inserting then
            RAISE_APPLICATION_ERROR(-20010, 'Inserting  Operation not allowed now');
         elsif deleting then
            RAISE_APPLICATION_ERROR(-20010, 'deleting  Operation not allowed now');
         else
            RAISE_APPLICATION_ERROR(-20010, 'updating  Operation not allowed now');
            end if;
      END IF;
   END;
   /

   BEGIN
      INSERT INTO test VALUES (2, 'amine', 'km', 500);

      delete from test;
   END;
   /
   begin
      update test
      set salary=500
      where id=1;
   end;
   /
   begin
      delete from test;
   end;
   /

-- Triggers row ------------------------------
 /*
      In Oracle, when working with triggers, the :NEW and :OLD qualifiers refer
      to the new and old values of the affected row in a table,
      depending on the type of trigger and the type ofoperation (INSERT, UPDATE, DELETE).
      These qualifiers are used within the body of a trigger
      to reference column values.
 */
 /*
   1-:NEW in INSERT Trigger:
      In the context of an INSERT trigger, :NEW represents the values that are being
      inserted into the table.
      You can reference :NEW.column_name to access the new values of specific columns.

   2-:OLD in DELETE Trigger:
      In the context of a DELETE trigger, :OLD represents the values that are being
      deleted from the table.
      You can reference :OLD.column_name to access the old values of specific columns.
      :NEW and :OLD in UPDATE Trigger:

   3-:NEW and :OLD in UPDATE Trigger:
      In the context of an UPDATE trigger, both :OLD and :NEW can be used.
      :OLD represents the old values before the update, and :NEW represents the new
      values after the update.
      You can compare the old and new values to perform actions based on changes.
 */
 --syntax :
   create or replace trigger triggerName
```

```
      before | after | instead of |
      insert | update | delete  -- you combine using or
      on tableName
      for each row
      begin
      --  code
      if inseting then -- check if the event is insert
      if deleting then -- check if the event is delete
      if updating then -- check if the event is update
      -- code
      end;

--* example 1 :
   CREATE OR REPLACE TRIGGER triggCheck_test_salary
   BEFORE
   UPDATE
   ON test
      for each row
      BEGIN
         if :new.salary <500 then
               raise_application_error(-20030,'min sal is 500');
               END IF;
      END;

   begin
      update test
      set salary=500
      where id=2;
      commit;
   end;
   /
--* example 2 :transaction history
  create table test_copy
  as select * from test;
   create table test_sal_audit
   (
   id number,
   OLD_sal number,
   NEW_sal number,
   op_date date,
   by_user varchar(30),
   description varchar2(100)
   );

   --- MY VERSION GET ME AN ERROR Failed:

   create or replace trigger trig_test_audit
   after
   insert or update or delete
   on test_copy
   for each row
   declare
   vcurrent_user VARCHAR2(30);
   begin
      select user into  vcurrent_user  from dual;
      if inserting then
```

```
2389          insert into  test_sal_audit
2390          values (:NEW.id,null,:NEW.salary,sysdate,vcurrent_user,'inserting');
2391       elsif deleting then
2392          insert into  test_sal_audit
2393          values (:OLD.id,:OLD.salary,NULL,sysdate,vcurrent_user,'deleting');
2394       else
2395          insert into  test_sal_audit
2396          values (:OLD.id,:OLD.salary,:NEW.salary,sysdate,vcurrent_user,'updating');
2397       end if;
2398    end;
2399
2400       BEGIN
2401          INSERT INTO  test_copy VALUES (2, 'amine', 'km', 500);
2402       END;
2403    rollback;
2404
2405    select * from test_sal_audit;
2406    select * from  test_copy;
2407
2408 -- to compile a triger :
2409    alter trigger triggerName compile;
2410
2411 -- to disable all trigers on a table :
2412    alter table tableName disable all triggers;
2413
2414 -- to enable all trigers on a table :
2415    alter table tableName enable all triggers;
2416
2417 -- to disable specific trigger :
2418    alter trigger triggerName disable ;
2419
2420 -- to enable specific trigger :
2421     alter trigger triggerName enable;
2422
2423 -- drop trigger :
2424       drop trigger triggerName;
2425
2426    --* example 3 ; default value
2427       create table customers
2428       (
2429          id number,
2430          name varchar(100),
2431          status char(1)
2432       );
2433       create or replace trigger triggDefaultCustom
2434       before insert
2435       on customers
2436       for each row
2437       when(new.name like 'A%')
2438       begin
2439          :new.status :='A';
2440       end;
2441
2442       begin
2443          insert into customers(id,name) values (1,'ayoub');
2444          insert into customers(id,name) values (1,'Ayoub');
```

```
      end;

      select * from customers;

-- triggers with view :
/*
   1- the new record 4 will not be inserted to the
      original
   Table customer, instead of that do other
       transactions.
   2- But you can still can insert to the original table but
        manually using eode

   when yo deal with view you mus update the table mainually
   insert
   delete update

*/
   create or replace view vEmp_all
    as select * from employees;

   create table dep_sal
   as
      select dep , sum(salary)  as sumEmp
      from  employees where dep is not null
      group by dep
      order by dep;

   CREATE OR REPLACE TRIGGER update_dep_sal
   INSTEAD OF INSERT OR DELETE
   ON vEmp_all
   declare
   vSal number default 0;
   BEGIN
      IF INSERTING THEN
         -- Insert into employees
         INSERT INTO employees
         VALUES (:NEW.employee_id, :NEW.first_name, :NEW.last_name, :NEW.hire_date,
COALESCE(:NEW.salary, 0), :NEW.dep);

         -- Update dep_sal
         UPDATE dep_sal
         SET sumEmp = sumEmp + COALESCE(:NEW.salary, 0)
         WHERE dep = :NEW.dep;
      ELSIF DELETING THEN
      delete from employees where employee_id=:old.employee_id;
         -- Update dep_sal
         UPDATE dep_sal
         SET sumEmp = sumEmp - COALESCE(:OLD.salary, 0)
         WHERE dep = :OLD.dep;
      else
      -- get the old salary of employee :
         select salary into vSal from employees
         where employee_id=:new.employee_id;

      --update salary
```

```
          update employees
          set salary = :new.salary
          where employee_id=:new.employee_id;

       -- update dep_sal :
          UPDATE dep_sal
          SET sumEmp = sumEmp - COALESCE(:new.salary, 0) + COALESCE(vSal, 0)
          WHERE dep = :OLD.dep;

       END IF;
    END;
    /


       insert into vEmp_all values (12,'km','tm',sysdate,500,1);

       update vEmp_all
       set salary =1000
       where employee_id=12;

       delete from vEmp_all where employee_id=12;

       delete from vEmp_all;
       select * from vEmp_all;
       select * from dep_sal ;

-- stop video 93 :
```