

## sumPlsql.sql

```
1  --Transaction: Propriétés
2      ACIDE :
3          Atomicité
4          Consistance
5          Isolation
6          Durabilité
7
8  ✦ Atomicité
9      /*L'ensemble des opérations d'une transaction apparaît comme une seule
10     opération atomique
11     Soit toutes les opérations sont validées ou toutes annulées (tout ou rien)
12     */
13
14  ✦ Consistance
15      /*L'exécution de la transaction fait passer la base de données d'un état
16     consistant à un autre état consistant*/
17
18  ✦ Isolation
19      /*Chaque transaction est indépendante des autres transactions concurrentes.
20     Sérialisation des transactions.
21     Les résultats d'une transaction ne sont visibles aux autres transactions
22     qu'une fois la transaction validée.
23     Les concurrences sont parfaitement contrôlées*/
24
25  ✦ Durabilité
26      /*C'est la persistance des mises à jour d'une transaction validée.
27     Les effets d'une transaction validée sont durables et permanents, quelques
28     soient les problèmes logiciels ou matériels, notamment après la fin de la
29     transaction.*/
30
31  -- base structure :
32      declare
33
34      begin
35
36      exceptions
37
38      end;
39      /
40
41  -- print :
42      dbms_output.put_line('your text ');
43
44  -- make the sql plus can print the result of a put_line function :
45      set serveroutput on;
46
47  -- declare a new variable
48      declare
49      variableName datatype
50
51  -- concatenation :
52      value1 || valu2
53
```

```

54  --* example 1 writing my age and full name on the screen :
55      set serveroutput on ;
56      declare
57      age number default 20;
58      fullName varchar(100) default 'ayoub majid ';
59      begin
60      dbms_output.put_line('the age is : ' || age);
61      dbms_output.put_line('the full name is : ' || fullName);
62      end;
63      /
64  -- assignment 1 :
65      -- write a bolck that outputs 'hello' and save it as test.sql
66      -- then run this script again
67  -- solution :
68      set serveroutput on;
69      begin
70      dbms_output.put_line('hello');
71      end;
72
73  -- call the file :
74      @ .\test.sql
75  --or
76      start .\test.sql
77
78  -- variables :
79      identifier [constant] datatype [not null] [:= | default expr];
80      -- WHEN you use no null you should give value :
81
82  --*example 1 :
83      declare
84      vDate date;
85      vNo number:=10;
86      vName varchar(100) not null:='ayoub';
87
88  --* example 2 :
89      declare
90      vDate date;
91      vNo number:=10;
92      vName varchar(100) not null:='ayoub';
93      begin
94      dbms_output.put_line('the date   : ' || vDate);
95      dbms_output.put_line('the number : ' || vNo);
96      dbms_output.put_line('the name   : ' || vName);
97
98      vNo:=vNo + 10;
99      dbms_output.put_line('the new  number : ' || vNo);
100
101      vName:='majid';
102      dbms_output.put_line('the new  name   : ' || vName);
103
104      vDate:='10-feb-2023';
105      dbms_output.put_line('the new  date   : ' || vDate);
106      end;
107
108  -- get the current date :
109      declare

```

```

110     vCurrrentDate date :=sysdate;
111
112 --* example 1 :
113     declare
114     vDate date :=sysdate;
115     vPi constant number:=3.14 ;
116     begin
117     vDate:= vDate + 10;
118     dbms_output.put_line('the current date date : ' || vDate);
119     dbms_output.put_line('the number : ' || vPi);
120     end;
121
122 -- q notation (to espace special characters)
123     q>('test');
124     --or
125     q['test'];
126     --or
127     q'x'test'x';
128 --*examaple 1:
129     begin
130     dbms_output.put_line(q'(father's day)');
131     end;
132 --*examaple 2:
133     begin
134     dbms_output.put_line( q'[father's day ]' );
135     end;
136 --*example 3:
137     select q'[today is the father's day not the mother's day ]'
138     from dual;
139
140 -- type of variables :
141 /*
142     Si une variable est déclarée avec l'option CONSTANTE, elle doit être initialisée
143     Si une variable est déclarée avec l'option NOT NULL, elle doit être initialisée
144     ✦NUMBER[(e,d)] Nombre réel avec e chiffres significatifs stockés et d décimales
145     ✦PLS_INTEGER Nombre entier compris entre -2 147 483 647 et +2 147 483 647
146     ✦CHAR [(n)] Chaîne de caractères de longueur fixe avec n compris entre 1 et 32767
147     ✦VARCHAR2[(n)] Chaîne de caractères de longueur variable avec n compris entre 1 et 32767
148     ✦BOOLEAN
149     ✦DATE
150     ✦RAW[[n)] Chaîne de caractères ou données binaires de longueur variable
151         avec n compris entre 1 et 32767. Le contenu
152         'une variable de ce type n'est pas interprété par PL/SQL
153     ✦LONG RAW Identique au type LONG qui peut contenir des données binaires
154     ✦LONG Chaîne de caractères de longueur variable avec au maximum 32760 octets
155     ✦ROWID Permet de stocker l'adresse absolue d'une ligne dans une table sous la
156         forme d'une chaîne de caractères '
157     */
158
159 -- interval year to month
160 --* example :
161     DECLARE
162     vName VARCHAR2(30) := 'ayoub';
163     trip INTERVAL YEAR TO MONTH := INTERVAL '1-4' YEAR TO MONTH;
164     BEGIN
165     DBMS_OUTPUT.PUT_LINE('My age is: ' || EXTRACT(YEAR FROM age) || ' years ' ||

```

```

166     EXTRACT(MONTH FROM age) || ' months');
167 END;
168 /
169
170 -- a type colonne:
171     nom_variable nom_table.nom_colonne%TYPE ;
172
173 -- b reference from another variable
174     nom_variable nom_variable_ref%TYPE ;
175
176 -- c type row :
177     nom_variable nom_table%ROWTYPE;
178
179 --* example 1 :
180     declare
181     vTest test%rowtype ;
182     begin
183     select * into vTest from test fetch first 1 row only ;
184     dbms_output.put_line('the first name is : ' || vTest.firstName);
185     dbms_output.put_line('the last name is : ' || vTest.lastName);
186     dbms_output.put_line('the the salary name is : ' || vTest.salary);
187     end;
188     /
189 --* example 2 :
190     set serveroutput on;
191     declare
192         vflag    boolean;
193         vno1     number default 21;
194         vno2     number default 20;
195         vprint   varchar(100);
196     begin
197         vflag:=false;
198         if vno1 =vno2 then
199             vflag:=true;
200             vprint:='numbers are equal';
201         else
202             vprint:='numbers are not equal';
203         end if;
204         -- print the stat
205         dbms_output.put_line(vprint);
206     end;
207     /
208
209 -- if structre :
210 -- if
211     IF condition THEN
212     instruction1 ;
213     instruction 2 ;
214     .....
215     instruction 2 ;
216     END IF;
217
218 -- if else
219     IF condition1 THEN
220     instruction1;
221     instruction 2;

```

```

222     ELSE
223         instruction3;
224     END IF;
225
226 -- if elsif esle
227     IF condition1 THEN
228         instruction1;
229         instruction 2;
230     ELSIF condition2 THEN
231         instruction 3;
232         instruction 4;
233     ELSIF condition3 THEN
234         instruction 5;
235         instruction 6;
236     ELSE instruction 7;
237     END IF;
238
239 -- relational operators :
240     =
241     <> or !=
242     >
243     <
244     >=
245     <=
246
247 -- logical operators :
248     and
249     or
250     not
251 -- example 1:
252     IF NOT (x = y) THEN
253         -- Code to execute if x is not equal to y
254     END IF;
255
256 -- bind variables delete
257     variable identifier [constant] datatype [not null] [:= | default expr];
258     /*
259     Bind variables are:
260     . Created in the environment
261     . Also called host variables
262     . Created with the VARIABLE keyword
263     . Used in SQL statements and PL/SQL blocks
264     . Accessed even after the PL/SQL block is executed
265     . Referenced with a preceding colon
266     */
267 -- composite data type :
268     TYPE nom_type_rec IS RECORD (
269         nom_champ1 type_élément1 [[ NOT NULL] := expression ],
270         nom_champ2 type_élément2 [[ NOT NULL] := expression ],
271         ...
272         nom_champN type_élémentN[[ NOT NULL] := expression ]
273     ) ;
274     Nom_variable nom_type_rec;
275
276 -- put value of select query in a variable :
277     select columnn 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 (through 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 through 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 --assignment 2 :
315     -- define a bind variable v_sal to be a number
316     -- create a block 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 -- writing 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

```

```

334 +      Addltron operator
335 -      Subtraction/negation operator
336 *      Multiplication operator
337 /      Division operator
338 =      Equality operator
339 @      Remote access indicator
340 ;      Statement temunator
341
342 -- Compound symbols
343 Symbol  -Meaning
344 <>      Inequality operator
345 !=      Inequality operator
346 ||      Concatenation operator
347 '---'   Single-li comment indicator
348 '/*'    Beginning comment delimiter
349 */      Ending comment delimiter
350 :=      Assignment operator
351
352 Data Type Conversion
353 • Converts data to comparable data types
354 • Is of two types:
355 Implicit conversion
356 Explicit conversion
357 • Functions:
358 - TO_char
359 - TO_date
360 - TO_number
361 - TO_timestamp
362
363 -- nested block :-----
364 -- outer block
365 declare
366 BEGIN
367     -- nseted block :
368     declare
369         begin
370             dbms_output.put_line('hello world');
371         end;
372 END;
373 /
374
375 --*example 1:
376 declare
377 vname varchar(30) := 'ayoub';
378 BEGIN
379     declare
380
381         begin
382             dbms_output.put_line('hello world my name is : '||vname );
383         end;
384 END;
385 /
386
387 --*example 2:
388 /*
389     access to global var from the nested block (in case when you have

```

```

390     a local and global var with the same name )
391 */
392 begin <<outer>>
393 declare
394     vname varchar(30) := 'ayoub';
395 BEGIN
396     declare
397     vname varchar(30) := 'kyoub';
398     begin
399         dbms_output.put_line('hello world my name is : '||outer.vname );
400         dbms_output.put_line('hello world my name is : '||vname );
401     end;
402 END;
403 end outer;
404 /
405 select * from test;
406
407 -- SQL Statements in PL/SQL
408     ✨ Retrieve a row from the database by using the SELECT
409     command.
410     ✨ Make changes to rows in the database by using DML
411     commands.
412     ✨ Control a transaction with the COMMIT, ROLLBACK, or
413     SAVEPOINT command.
414     ✨ PL/SQL does not directly support data definition language (DDL) statements,
415     ✨ PL/SQL does not directly support data control language (DCL) statements,
416     such as GRANT or REVOKE. You can use dynamic SQL to execute them.
417
418 -- SQL Cursor
419 /*
420  . A cursor is a pointer to the private memory area allocated by
421  the Oracle server.
422  . A cursor is used to handle the result set of a SELECT
423  statement.
424  . There are two types of cursors:
425      ✨ - Implicit: Created and managed internally by the Oracle
426      server to process SQL statements
427      ✨ - Explicit: Declared explicitly by the programmer
428
429      SQL Cursor Attributes for Implicit Cursors
430
431      Using SQL cursor attributes, you can test the outcome of your
432      SQL statements.
433  */
434 -- SQL%FOUND
435 Boolean attribute that evaluates to TRUE if the
436 most recent SQL statement returned at least one
437 row
438
439 -- SQL%NOTFOUND
440 Boolean attribute that evaluates to TRUE if
441 the most recent SQL statement did not
442 return even one row
443
444 -- SQL%ROWCOUNT
445 --is only accurate after a statement that modifies data (e.g., INSERT, UPDATE, DELETE)

```



```

446 An integer value that represents the number of
447 rows affected by the most recent SQL statement
448
449 -- example rowcount :
450 begin
451 update test set salary =0 where id=1;
452 dbms_output.put_line('the number of rows updated is : ' || sql%rowcount);
453 end;
454
455 -- example found :
456 declare
457 vrowsExist boolean default false;
458 begin
459 update test set salary =0 where id=10;
460 vrowsExist:= sql%found;
461 if vrowsExist then
462 dbms_output.put_line('the stat : yes');
463 else
464 dbms_output.put_line('the stat : no');
465 end if;
466 end;
467 /
468
469 -- change the prompt message -----[-]:
470 you can change the prompt as follow
471 but it should executed as a script
472 syntaxe :
473 ACCEPT Variable_name PROMPT 'Messgae' -- accept =define + prompt change message
474 -- this message it will just assoscited with only this variable
475
476 --* example 1:
477 ACCEPT User_id PROMPT 'Pelase enter the user id :';
478 select employee_id,first_name,last_name,salary
479 from employees
480 where employee_id=&User_id;
481
482 --* example 2:
483 select first_name,last_name,&&User_column
484 from employees
485
486 order by &User_column;
487 --* example 3:
488 ACCEPT User_column PROMPT 'Pelase enter the Column :';
489 select first_name,last_name,&User_column
490 from employees
491 order by &User_column;
492
493
494 -- case expression :
495 case selector
496 when expression then result1
497 when expression then result2
498 ...
499 when expression then resultN
500 [else resultN+1]
501 end;

```

```

502 /
503 -- case statements :
504 case
505 when condition1 then
506 -- code
507 when condition2 then
508 -- code
509 ...
510 when conditionN then
511 -- code
512 [else
513 -- code ]
514 end case;
515 /
516
517 --* example 1 :
518 select firstname,lastname ,length(firstname) ,case length(firstname)
519 when 4 then '4 char '
520 when 5 then '5 char '
521 when 6 then '6 char'
522 else 'n/a'
523 end
524 from test;
525
526 --* example 2 :
527 select firstname,lastname ,length(firstname) ,case
528 when length(firstname) =4 then '4 char '
529 when length(firstname)=5 then '5 char '
530 when length(firstname) =6 then '6 char'
531 else 'n/a'
532 end
533 from test;
534
535 --* example 3 :
536 ACCEPT empId PROMPT 'Enter the employee id: ';
537 DECLARE
538     vSal NUMBER;
539     vDesc VARCHAR(100);
540 BEGIN
541     SELECT salary INTO vSal FROM test WHERE id = &empId;
542
543     vDesc := CASE
544         WHEN vSal IS NULL THEN 'No salary for the employee'
545         WHEN vSal BETWEEN 1000 AND 3000 THEN 'Salary is low'
546         WHEN vSal BETWEEN 3001 AND 5000 THEN 'Salary is medium'
547         WHEN vSal BETWEEN 5001 AND 10000 THEN 'Salary is good'
548         ELSE 'Salary is good'
549     END;
550
551     DBMS_OUTPUT.PUT_LINE('The employee status: ' || vDesc);
552 END;
553 /
554
555 --* example 3 : case statements
556 ACCEPT empId PROMPT 'Enter the employee id: ';
557 DECLARE

```

```

558     vSal NUMBER;
559     vDesc VARCHAR(100);
560 BEGIN
561     SELECT salary INTO vSal FROM test WHERE id = &empId;
562
563 CASE
564     WHEN vSal IS NULL THEN
565         DBMS_OUTPUT.PUT_LINE('No salary for the employee');
566     WHEN vSal BETWEEN 1000 AND 3000 THEN
567         DBMS_OUTPUT.PUT_LINE('Salary is low');
568     WHEN vSal BETWEEN 3001 AND 5000 THEN
569         DBMS_OUTPUT.PUT_LINE('Salary is medium');
570     WHEN vSal BETWEEN 5001 AND 10000 THEN
571         DBMS_OUTPUT.PUT_LINE('Salary is good');
572     ELSE
573         DBMS_OUTPUT.PUT_LINE('Salary is good');
574 END case;
575 END;
576 /
577
578 -- handling nulls value
579 -- any comparaisn throught if with null      :
580 /*
581 the if block :we not execut
582 */
583
584 -- solution
585 nvl(var,backValue)
586
587 --* example 1 :
588 declare
589 x number default 2;
590 y number default null;
591 begin
592     if nvl(x,0) <> nvl(y,0) then
593         dbms_output.put_line('hi');
594     end if;
595 end;
596 /
597 -- • Loops repeat a statement (or sequence of statements)
598 multiple times.
599 <<label>>
600 LOOP
601     instruction1;
602     instruction2;
603     EXIT [label][WHEN condition1];
604 END LOOP label;
605
606 •EXIT force la sortie de la boucle sans conditions.
607 •EXIT WHEN permet une sortie de boucle si la condition est vraie.
608 •EXIT <<label>> WHEN permet une sortie d'une boucle nommée label si la condition
609 est vraie.
610 •EXIT <<label>> force une sortie de boucle nommée label.'
611
612 --* example 1: print value from 0 to 10:
613 loop

```

```

614         i:=i+1;
615         dbms_output.put_line(i);
616         exit when i=10;
617     end loop;
618 end;
619
620 --* example 2 : multiplication table from 1 to 2 :
621 set verify off;
622 declare
623     i integer:=0; j integer:=0;
624 begin
625     loop
626     i:=i+1;j:=0;
627         loop
628             j:=j+1;
629             dbms_output.put_line(i || '*' || j || '=' || i*j);
630             exit when j=10;
631         end loop;
632         dbms_output.put_line(' ');
633     exit when i=2;
634 end loop;
635 end;
636 /
637
638 --* example 3: of exit in the first iteration : print multiplication talbe of 1
639 set verify off;
640 declare
641     i integer:=0;
642     j integer:=0;
643 begin
644     <<lable1>>
645     loop
646     i:=i+1;
647     j:=0;
648     <<lable2>>
649         loop
650             j:=j+1;
651             dbms_output.put_line(i || '*' || j || '=' || i*j);
652             exit lable1 when j=10;
653         end loop lable2;
654         dbms_output.put_line(' ');
655     exit when i=2;
656 end loop lable1;
657 end;
658 /
659
660 --* example 4 : get the first name of employee 8 9 10:
661 declare
662     vfirstName employees.first_name%type;
663     vcounter integer :=7;
664 begin
665     loop
666         vcounter:= vcounter + 1;
667         select first_name into vfirstName from employees where employee_id=vcounter;
668         dbms_output.put_line('the first name of employee : ' || vfirstName);
669         exit when vcounter=10;

```

```

670     end loop;
671     end;
672     /
673
674 -- while Loop:
675     WHILE conditions
676     LOOP
677         instruction1;
678         instruction2;
679     END LOOP;
680 --* example 1: print hello wolrd three times:
681     declare
682         vcounter integer :=1;
683     begin
684         while vcounter <= 3 loop
685             dbms_output.put_line(vcounter || '-hello wolrd');
686             vcounter:= vcounter +1;
687         end loop;
688     end;
689     /
690
691 --* example 2: print numbers from 0 to 10;
692     while i<10 loop
693         dbms_output.put_line(i);
694         i:=i+1;
695     end loop;
696
697 --* example 3: -- get the first name of employee 8 9 10:
698     declare
699         vfirstName employees.first_name%type;
700         vcounter integer :=8;
701     begin
702         while vcounter <= 10 loop
703             select first_name into vfirstName from employees
704             where employee_id= vcounter;
705             dbms_output.put_line(' the firstName : ' || vfirstName);
706             vcounter := vcounter +1;
707         end loop;
708     end;
709     /
710
711 -- for in Loop:
712     FOR compteur IN [REVERSE] borne_inf..borne_sup LOOP
713         instruction1 ;
714         instruction2 ;
715         instruction3 ;
716         [EXIT WHEN condition];
717     END LOOP;
718
719 --* example 1 : print numbers from 1 to 5
720     for i in 1..5 loop
721         dbms_output.put_line(i);
722     end loop;
723
724 --* example 2 : print numbers from 5 to 1 :
725     for i in reverse 1..5 loop

```

```

726     dbms_output.put_line(i);
727     end loop;
728
729
730 --* example 3 : show just even numbers from 1 to 10 :
731     DECLARE
732     BEGIN
733         FOR i IN 1..10 LOOP
734             IF MOD(i, 2) = 1 THEN
735                 -- Skip odd numbers
736                 CONTINUE;
737             END IF;
738
739             DBMS_OUTPUT.PUT_LINE(i);
740         END LOOP;
741     END;
742 /
743 --* example 3 :
744     FOR client_rec IN (SELECT NO, NOM, VILLE FROM E_CLIENT) LOOP
745         DBMS_OUTPUT.PUT_LINE('Client Number: ' || client_rec.NO || ', Name: '
746             || client_rec.NOM || ', City: ' || client_rec.VILLE);
747     END LOOP;
748
749 --* example 4 : print triangle :
750     SET VERIFY OFF;
751     ACCEPT inpn PROMPT 'Enter the size of triangle: ';
752     DECLARE
753         vn INTEGER := &inpn;
754     BEGIN
755         IF vn > 30 THEN
756             DBMS_OUTPUT.PUT_LINE('The available size is 30');
757         ELSE
758             FOR i IN 1..vn LOOP
759                 FOR j IN 1..i LOOP
760                     DBMS_OUTPUT.PUT('*');
761                 END LOOP;
762                 DBMS_OUTPUT.PUT_LINE('');
763             END LOOP;
764         END IF;
765     END;
766 /
767
768 -- write a program to print :
769 /*
770     1
771     :)
772     2
773     :)
774     3
775     :)
776     4
777     :)
778     5
779     :)
780     6
781     7

```

```

782      8
783      9
784     10
785 */
786 --* method 1 :
787     dbms_output.put_line( chr(10) || 'first method : ');
788     for i in 1..10 loop
789         dbms_output.put_line(chr(9) || i);
790         if i>=6 then
791             continue;
792         end if;
793         dbms_output.put_line(chr(9) || ':');
794     end loop;
795
796 --* method 2 :
797     dbms_output.put_line( chr(10) || 'second method : ');
798     for i in 1..10 loop
799         dbms_output.put_line(chr(9) || i);
800         if i<=5 then
801             dbms_output.put_line(chr(9) || ':');
802         end if;
803     end loop;
804
805 -----
806 Composite Data Types
807     • Can hold multiple values (unlike scalar types)
808     • Are of two types:
809         - PL/SQL records
810         - PL/SQL collections
811             - INDEX BY tables or associative arrays
812             - Nested table
813             - VARRAY
814
815 Declaring a PL/SQL Record
816     1- programmer-defined records.
817     2- table-based record. %Rowtype
818     3- cursor-based record. ( will be covered later )
819
820 -- 1- programmer-defined records :
821     TYPE nom_type_rec IS RECORD (
822         nom_champ1 type_élément1 [[ NOT NULL] := expression ],
823         nom_champ2 type_élément2 [[ NOT NULL] := expression ],
824         ...
825         nom_champN type_élémentN[[ NOT NULL] := expression ]
826     ) ;
827 --* example 1:
828     declare
829         type stemp is record (
830             vempid employees.employee_id%type,
831             vfirstname employees.first_name%type,
832             vlastname employees.last_name%type
833         );
834         vemp stemp;
835     begin
836         select
837             employee_id,

```

```

838         first_name,
839         last_name into vemp
840     from
841         employees
842     where
843         employee_id=1;
844     dbms_output.put_line('the id : ' || vemp.vempid);
845     dbms_output.put_line('the first name : ' || vemp.vfirstname);
846     dbms_output.put_line('the last name : ' || vemp.vlastname);
847 end;
848 /
849
850 -- create an empty copy from an existant table :
851 -- (notice : the copy table doesn't copy constraints excpet not null)
852 create table copyTableName as select columns from mainTableName where 1=2;
853
854 -- example 1: insert row to copyTable using a record
855 create table copyEmp
856 as select employee_id ,first_name,last_name from employees
857 where 1=2;
858 declare
859     type stemp is record (
860         vempid employees.employee_id%type,
861         vfirstname employees.first_name%type,
862         vlastname employees.last_name%type
863     );
864     vemp stemp;
865 begin
866     select employee_id, first_name, last_name into vemp from employees
867     where employee_id=1;
868
869     insert into copyEmp values vemp;
870 end;
871 /
872 select * from copyEmp;
873
874
875 -- 2- table-based record. %Rowtype
876 VarName tableName%rowtype;
877
878 --* example 1: insert row to copyTable using a record
879 create table copyEmp
880 as select * from employees
881 where 1=2;
882 declare
883     vemp employees%rowtype;
884 begin
885     select * into vemp from employees
886     where employee_id=1;
887     insert into copyEmp values vemp;
888 end;
889 /
890 select * from copyEmp;
891
892
893 -- update row directly :

```



```

894     update tableName;
895     set row=stTabName;
896
897 --*exmaple 1 :
898     declare
899         vemp employees%rowtype;
900     begin
901         vemp.employee_id :=10;
902         vemp.first_name :='ayoub';
903         update copyEmp
904         set row=vemp;
905     end;
906     /
907     select * from copyEmp;
908
909 -- PL/SQL collections -----
910
911     - INDEX BY tables or associative arrays
912     - Nested table
913     - VARRAY
914
915 -- INDEX BY Tables or Associative Arrays
916 /*
917  . Are PL/SQL structures with two columns:
918      - Primary key of integer or string data type
919      - Column of scalar or record data type
920  . Are unconstrained in size. However, the size depends on the
921    values that the key data type can hold.
922
923 */
924
925 -- syntax :
926     type tableName is table of valueType
927     index by [pls_integer | binary_integer | varchar2(size)];
928 -- set data :
929     vtablename(key):=value;
930 -- get data :
931     vtableName(key);
932
933 --example 1 :
934     declare
935         type arrtab is table of varchar2(100)
936         index by pls_integer;
937         vtab arrtab;
938     begin
939         vtab(-1):='KMoub';
940         vtab(0):='KMoub';
941         vtab(1):='ayoub';
942         vtab(2):='amine';
943         dbms_output.put_line(vtab(-1));
944         dbms_output.put_line(vtab(0));
945         dbms_output.put_line(vtab(1));
946         dbms_output.put_line(vtab(2));
947     end;
948     /
949 -- table methods :

```

```

950 vtablename.method([parameters]);
951 -- EXISTS (n)
952   .Returns TRUE if the nth element in a PL/SQL table exists
953
954 -- COUNT
955   .Returns the number of elements that a PL/SQL table currently
956
957 -- FIRST
958   . Returns the first (smallest) index number in a PL/SQL table
959   . Returns NULL if the PL/SQL table is empty
960
961 -- LAST
962   . Returns the last (largest) index number in a PL/SQL table
963   . Returns NULL if the PL/SQL table is empty
964
965 -- PRIOR (n)
966   .Returns the index number that precedes index n in a PL/SQL table
967
968 -- NEXT (n)
969   .Returns the index number that succeeds index n in a PL/SQL table
970
971 -- DELETE
972   .DELETE removes all elements from a PL/SQL table.
973   .DELETE (n) removes the nth element from a PL/SQL table.
974   .DELETE (m, n) removes all elements in the range m ... n from a pl/sql table
975
976 --* example 1:
977 declare
978 type arrtab is table of varchar2(100)
979 index by pls_integer;
980 vtab arrtab;
981 begin
982   vtab(1):='KMoub';
983   vtab(2):='KMoub';
984   vtab(6):='KMoub';
985   vtab(9):='ayoub';
986   for i in 1..10 loop
987     if vtab.exists(i) then
988       dbms_output.put_line( chr(9)|| i || '-element exist : ' || vtab(i));
989     else
990       dbms_output.put_line( chr(9)|| i || '-element not exist : -) ');
991     end if;
992   end loop ;
993   dbms_output.put_line('the total number of elements : ' || vtab.count());
994   dbms_output.put_line('the first element : ' || vtab.first());
995   dbms_output.put_line('the next element index after index 2 : ' || vtab.next(2));
996 end;
997 /
998
999 --* example 2 : rowtype
1000 declare
1001 type arrtab is table of employees%rowtype
1002 index by pls_integer;
1003 vtab arrtab;
1004 begin
1005   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

```

```

1062 -- increase the size by one :
1063 arrName.extend;
1064 -- example :
1065 declare
1066 type arrlocation is table of varchar2(100);
1067 loc arrlocation;
1068
1069 begin
1070
1071     loc:=arrlocation('Morocco','Jordan','UKA');
1072     loc.extend;
1073     loc(4):='uk';
1074     dbms_output.put_line(loc(1));
1075     dbms_output.put_line(loc(2));
1076     dbms_output.put_line(loc(3));
1077     dbms_output.put_line(loc(4));
1078 end;
1079 /
1080
1081 -- delete element :
1082 arrName.delete(index);
1083 --example :
1084 declare
1085 type arrlocation is table of varchar2(100);
1086 loc arrlocation;
1087
1088 begin
1089     loc:=arrlocation('Morocco','Jordan','UKA');
1090     loc.extend;
1091     loc.delete(1);
1092     dbms_output.put_line(loc(2));
1093     dbms_output.put_line(loc(3));
1094 end;
1095 /
1096
1097 -- VARRAY : like nested array but with fixed size
1098 --syntax
1099 type tableName is varray(size) of valueType
1100
1101 --*example 1:
1102 declare
1103 type arrlocation is varray(3) of varchar2(100);
1104 loc arrlocation;
1105
1106 begin
1107     -- you should initialise it :
1108     loc:=arrlocation('Morocco','Jordan','UKA');
1109
1110     dbms_output.put_line(loc(1));
1111     dbms_output.put_line(loc(2));
1112     dbms_output.put_line(loc(3));
1113 end;
1114 /
1115
1116
1117 -- pop element :

```

```

1118         arrName.trim(nbrOfElemels);
1119     --*examaple :
1120     declare
1121         type arrlocation is varray(3) of varchar2(100);
1122         loc arrlocation;
1123
1124     begin
1125         -- you should initialise it :
1126         loc:=arrlocation('Morocco','Jordan','UKA');
1127         loc.trim(2);
1128
1129         dbms_output.put_line(loc(1));
1130
1131     end;
1132     /
1133
1134
1135 -- using Explicit cursor -----
1136 /*
1137     Every SQL statement executed by the Oracle server has an
1138     associated individual cursor:
1139     . Implicit cursors: Declared and managed by PL/SQL for all
1140     DML and PL/SQL SELECT statements
1141     . Explicit cursors: Declared and managed by the programmer
1142 */
1143 -- Explicit Cursor Operations
1144 /*
1145     You declare explicit cursors in PL/SQL when you have a SELECT statement that returns
multiple
1146     rows. You can process each row returned by the SELECT statement.
1147 */
1148 --Explicit cursor functions:
1149 /*
1150     . Can perform row-by-row processing beyond the first row returned by a query
1151     . Keep track of the row that is currently being processed
1152     . Enable the programmer to manually control explicit cursors in the PL/SQL block
1153 */
1154
1155 --define a cursor :
1156     cursor cursorName is (select columns from tableName [where ...]);
1157
1158 -- open a cursor :
1159     open cursorName;
1160     /*
1161         The OPEN statement executes the query associated with the cursor, identifies the active
set, and
1162         positions the cursor pointer at the first row. The OPEN statement is included in the
executable section
1163         of the PL/SQL block.
1164         OPEN is an executable statement that performs the following operations:
1165         1. Dynamically allocates memory for a context area
1166         2. Parses the SELECT statement
1167         3. Binds the input variables (sets the values for the input variables by obtaining their
memory
1168         addresses)
1169         4. Identifies the active set (the set of rows that satisfy the search criteria). Rows in
the active

```

```

1170     set are not retrieved into variables when the OPEN statement is executed. Rather, the
1171     FETCH statement
1172     retrieves the rows from the cursor to the variables.
1173     5. Positions the pointer to the first row in the active set
1174     Note: If a query returns no rows when the cursor is opened, PL/SQL does not raise an
1175     exception.
1176     You can find out the number of rows returned with an explicit cursor by using the
1177     <cursor name>%ROWCOUNT attribute.
1178     */
1179
1180 -- Fetching Data from the Cursor
1181 fetch cursorName into vDatName;
1182 /*
1183     The FETCH statement retrieves the rows from the cursor one at a time. After each fetch,
1184     the cursor
1185     advances to the next row in the active set. You can use the %NOTFOUND attribute to
1186     determine
1187     whether the entire active set has been retrieved.
1188
1189     The FETCH statement performs the following operations:
1190     1. Reads the data for the current row into the output PL/SQL variables
1191     2. Advances the pointer to the next row in the active set
1192     */
1193
1194 -- Closing the Cursor
1195 close cursorName;
1196 /*
1197     The CLOSE statement disables the cursor, releases the context area, and "undefines" the
1198     active set.
1199     Close the cursor after completing the processing of the FETCH statement. You can reopen
1200     the cursor
1201     if required. A cursor can be reopened only if it is closed. If you attempt to fetch data
1202     from a cursor
1203     after it has been closed, then an INVALID_CURSOR exception will be raised.
1204     Note: Although it is possible to terminate the PL/SQL block without closing cursors, you
1205     should
1206     make it a habit to close any cursor that you declare explicitly to free up resources.
1207     There is a maximum limit on the number of open cursors per session, which is determined
1208     by the
1209     OPEN_CURSORS parameter in the database parameter file. (OPEN_CURSORS = 50 by default.)
1210     */
1211
1212 -- cursorname%ISOPEN 1
1213     Evaluates to TRUE if the cursor is open
1214
1215 -- cursorname%FOUND
1216     Boolean attribute that evaluates to TRUE if the
1217     most recent SQL statement returned at least one
1218     row
1219
1220 -- cursorname%NOTFOUND
1221     Boolean attribute that evaluates to TRUE if
1222     the most recent SQL statement did not
1223     return even one row
1224
1225 -- cursorname%ROWCOUNT
1226     --is only accurate after a statement that modifies data (e.g., INSERT, UPDATE, DELETE)
1227     An integer value that represents the number of

```

```

1219 rows affected by the most recent SQL statement
1220
1221 --* example 1 :
1222 set serveroutput on;
1223 declare
1224     cursor cEmp is
1225     select * from employees where employee_id between 1 and 6;
1226     vEmp employees%rowtype;
1227 begin
1228     -- open the cursor :
1229     open cEmp;
1230
1231     dbms_output.put_line(chr(10)|| ' ----employees dat from 1 to 6---- ' || chr(10));
1232     -- fetch the first row from the cursor :
1233     fetch cEmp into vEmp;
1234
1235     while cEmp%found loop
1236         -- print the data of employee :
1237         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1238         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1239         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1240         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary || chr(10));
1241
1242         -- fetch again till the end :
1243         fetch cEmp into vEmp;
1244     end loop;
1245     -- close the cursor :
1246     close cEmp;
1247 end;
1248 /
1249
1250 -- define a variable of cursor :
1251 declare
1252     varNameee cursorname%rowtype;
1253
1254 --* example 2 : using for loop
1255 /*
1256 1-cursor open automatically
1257 3- auto fetch
1258 2-cursor close automatically
1259 */
1260 set serveroutput on;
1261 declare
1262     cursor cEmp is
1263     select * from employees where employee_id between 1 and 6;
1264 begin
1265
1266     dbms_output.put_line(chr(10)|| ' ---- employees dat from 1 to 6---- ' || chr(10));
1267
1268     for vEmp in cEmp loop
1269         -- print the data of employee :
1270         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1271         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1272         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1273         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary || chr(10));
1274     end loop;

```

```

1275     end;
1276     /
1277
1278 --* exmaple 3 : increase the salary by 100$ for employees where the id between 1 and 6
1279     select * from employees
1280     where employee_id between 1 and 6;
1281     set serveroutput on;
1282     declare
1283         cursor cEmp is
1284         select * from employees where employee_id between 1 and 6;
1285     begin
1286         dbms_output.put_line(chr(10)|| ' ---- update employees dat from 1 to 6---- ' || chr(10));
1287
1288         for vEmp in cEmp loop
1289             update employees
1290             set salary=salary +100
1291             where employee_id=vEmp.employee_id;
1292         end loop;
1293         commit;
1294     end;
1295     /
1296     select * from employees
1297     where employee_id between 1 and 6;
1298
1299 --*example 4 :
1300     set serveroutput on;
1301     declare
1302         cursor cEmp is
1303         select * from employees;
1304         vEmp cEmp%rowtype;
1305     begin
1306         if cEmp%isopen then
1307             null;
1308         else
1309             open cEmp;
1310         end if;
1311         dbms_output.put_line('the counter for cursor now is : ' || cEmp%rowcount);
1312         loop
1313             fetch cEmp into vEmp;
1314             exit when cEmp%notfound or cEmp%rowcount >6 ;
1315             dbms_output.put_line('the counter for cursor now is : ' || cEmp%rowcount);
1316         end loop;
1317     end;
1318     /
1319
1320 -- cursor with parameters :
1321     cursor cursorName(arg1 datatype ...) is (select columns from tableName [where ...]);
1322     /*
1323
1324     Parameter data types are the same as those for scalar variables, but you do not give them
1325     sizes.
1326
1327     You can pass parameters to the cursor that is used in a cursor FOR loop:
1328     DECLARE
1329     CURSOR c_emp_cursor (p_deptno NUMBER, p_job VARCHAR2) IS
1330     SELECT

```



```

1330 BEGIN
1331 FOR emp_record IN c_emp_cursor (10, 'Sales') LOOP ...
1332 */
1333 ...
1334
1335 --* example 1: Loop
1336 set serveroutput on;
1337 declare
1338     cursor cEmp(vDep number) is
1339     select * from employees where dep=vdep;
1340     vEmp cEmp%rowtype;
1341 begin
1342     open cEmp(1);
1343     dbms_output.put_line(chr(10)|| ' ---- dep 1 --- ' || chr(10));
1344     loop
1345         fetch cEmp into vEmp;
1346         exit when cEmp%notfound or cEmp%rowcount >6 ;
1347         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1348         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1349         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1350         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary );
1351         dbms_output.put_line(chr(9) || 'the departement : ' || vEmp.dep || chr(10));
1352     end loop;
1353     close cEmp;
1354
1355     open cEmp(2);
1356     dbms_output.put_line(chr(10)|| ' ---- dep 2 --- ' || chr(10));
1357     loop
1358         fetch cEmp into vEmp;
1359         exit when cEmp%notfound or cEmp%rowcount >6 ;
1360         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1361         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1362         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1363         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary );
1364         dbms_output.put_line(chr(9) || 'the departement : ' || vEmp.dep || chr(10));
1365     end loop;
1366     close cEmp;
1367
1368     open cEmp(3);
1369     dbms_output.put_line(chr(10)|| ' ---- dep 3 --- ' || chr(10));
1370     loop
1371         fetch cEmp into vEmp;
1372         exit when cEmp%notfound or cEmp%rowcount >6 ;
1373         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1374         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1375         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1376         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary );
1377         dbms_output.put_line(chr(9) || 'the departement : ' || vEmp.dep || chr(10));
1378     end loop;
1379     close cEmp;
1380
1381 end;
1382 /
1383
1384 --*example 2 : for Loop
1385 set serveroutput on;

```

```

1386 declare
1387     cursor cEmp(vDep number) is
1388         select * from employees where dep=vdep;
1389         vEmp cEmp%rowtype;
1390 begin
1391     dbms_output.put_line(chr(10)|| ' ---- dep 1 --- ' || chr(10));
1392     for vEmp in cEmp(1)
1393     loop
1394         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1395         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1396         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1397         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary );
1398         dbms_output.put_line(chr(9) || 'the departemnt : ' || vEmp.dep || chr(10));
1399     end loop;
1400
1401
1402     dbms_output.put_line(chr(10)|| ' ---- dep 2 --- ' || chr(10));
1403     for vEmp in cEmp(2)
1404     loop
1405         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1406         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1407         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1408         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary );
1409         dbms_output.put_line(chr(9) || 'the departemnt : ' || vEmp.dep || chr(10));
1410     end loop;
1411
1412     dbms_output.put_line(chr(10)|| ' ---- dep 3 --- ' || chr(10));
1413     for vEmp in cEmp(3)
1414     loop
1415         dbms_output.put_line(chr(9) || 'the id : ' || vEmp.employee_id);
1416         dbms_output.put_line(chr(9) || 'the first name : ' || vEmp.first_name);
1417         dbms_output.put_line(chr(9) || 'the last name : ' || vEmp.last_name);
1418         dbms_output.put_line(chr(9) || 'the salary : ' || vEmp.salary );
1419         dbms_output.put_line(chr(9) || 'the departemnt : ' || vEmp.dep || chr(10));
1420     end loop;
1421
1422 end;
1423 /
1424
1425
1426 -- for update clause :
1427     cursor cursorName is (select columns from tableName [where ...])
1428     for update [of column1,column2... | nowait | wait duration];
1429 /*
1430     .nowait : don't wait just return an error
1431     .wait 30 : return error after 30 second else execute
1432     . Use explicit locking to deny access to other sessions for the
1433     duration of a transaction.(unti commit or rollback)
1434     . Lock the rows before the update or delete.
1435 */
1436
1437 -- use it rather than : where table_id=record.id;
1438     where current of cursorName;
1439
1440 --*example 1 :
1441     select * from employees where dep in (2,3);

```

```

1442 set serveroutput on;
1443 declare
1444     cursor cEmp is
1445         select * from employees where dep in (2,3) for update of salary;
1446         vEmp cEmp%rowtype;
1447 begin
1448
1449     dbms_output.put_line(chr(10)|| ' ---- dep 1 & 2 --- ' || chr(10));
1450     for vEmp in cEmp
1451     loop
1452         update employees
1453         set salary= salary -100
1454         -- use it rather than : where employee_id=vEmp.employee_id;
1455         where current of cEmp;
1456     end loop;
1457     commit;
1458 end;
1459 /
1460 select * from employees where dep in (2,3);
1461
1462
1463
1464 -- handling exception -----
1465 /*
1466 . An exception is a PL/SQL error that is raised during program
1467   execution.
1468 . An exception can be raised:
1469   - Implicitly by the Oracle server
1470   - Explicitly by the program
1471 . An exception can be handled:
1472   - By trapping it with a handler
1473   - By propagating it to the calling environment
1474 */
1475
1476 -- syntax :
1477 EXCEPTION
1478 WHEN exception1 [OR exception2 . . .] THEN
1479     statement1;
1480     statement2;
1481
1482 [WHEN exception3 [OR exception4 . . .] THEN
1483     statement1;
1484     statement2;
1485 .]
1486 [WHEN OTHERS THEN
1487     statement1;
1488     statement2;
1489 . . .]
1490
1491 /*
1492 . The EXCEPTION keyword starts the exception-handling
1493   section.
1494 . Several exception handlers are allowed.
1495 . Only one handler is processed before leaving the block.
1496 . WHEN OTHERS is the last clause.
1497

```

```

1498 */
1499 --most common defined exceptions
1500 /*
1501  . Reference the predefined name in the exception-handling
1502  routine.
1503  . Sample predefined exceptions:
1504  - NO_DATA_FOUND
1505  - TOO_MANY_ROWS
1506  - INVALID_CURSOR
1507  - ZERO_DIVIDE
1508  - DUP_VAL_ON_INDEX (insert with duplicate primary key)
1509 */
1510
1511 --* example 1 :
1512 declare
1513     vLastname varchar(30);
1514 begin
1515     select last_name into vLastname from employees
1516     where employee_id=12/0;
1517     dbms_output.put_line('the last name : ' || vLastname);
1518
1519     exception
1520     when no_data_found then
1521         dbms_output.put_line('the query doesn't retrieve any record ');
1522     when too_many_rows then
1523         dbms_output.put_line('the query retrieve more than one record ');
1524     when zero_divide then
1525         dbms_output.put_line('divide by 0 not allowed');
1526     when others then
1527         dbms_output.put_line('other Error');
1528 end;
1529 /
1530
1531 --* example 2 :
1532 declare
1533     vfirstName employees.first_name%type;
1534 begin
1535     for i in 5..12 loop
1536         begin
1537             select first_name into vfirstName from employees
1538             where employee_id=i;
1539             dbms_output.put_line('the first name : ' || vfirstname);
1540
1541             exception
1542             when no_data_found then
1543                 null;
1544         end;
1545     end loop;
1546 exception
1547     when no_data_found then
1548         null;
1549 end;
1550 /
1551
1552 -- declare exception :
1553 vexpceptionName exception ;

```

```

1554
1555 -- add code to expcetion :
1556     program 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 ----
1622 -- raise exception :
1623     raise vexceptionName;
1624
1625 -- raise exception without define exception section :
1626 /*
1627     Is a user-specified number for the exception between -20.000
1628     and -20.999
1629     Is the user-specified message for the exception; is a character string
1630     up to 2.048 bytes long
1631
1632     Is an optional Boolean parameter (If TRUE, the error is placed
1633     on the stack of previous errors. If FALSE, which is the default, the
1634     error replaces all previous errors.)
1635 */
1636 raise_application_error(errorCode,'msg');
1637
1638 --* example 1:
1639 declare
1640     vempId number := 11;
1641     expinvalId exception ;
1642 begin
1643     update employees
1644     set salary =2000
1645     where employee_id =vempId;
1646
1647     if sql%notfound then
1648         raise expinvalId;
1649     end if;
1650
1651     exception
1652     when expinvalId then
1653         dbms_output.put_line('id not found ');
1654 end;
1655 /
1656
1657 --* example 2 :
1658 declare
1659     vempId number := 11;
1660 begin
1661     update employees
1662     set salary =2000
1663     where employee_id =vempId;
1664
1665     if sql%notfound then

```

```

1665         raise_application_error(-20000,'id not found ');
1666     end if;
1667 end;
1668 /
1669
1670 -- procedure : -----
1671 -- syntax :
1672 create or replace procedure procedureName
1673 [(par1 in datatype ... )] --parameters without size :
1674 is
1675     --declare variables
1676 begin
1677     [excpetion]
1678
1679 end;
1680
1681 -- call a procedure :
1682 --outside the plsql block (begin end ) :
1683     execute procedureName(parametersValues);
1684 -- in plsql block :
1685     procedureName(parametersValues);
1686
1687 --* example 1 : create a procedure that print hello world
1688 -- create hello procedure :
1689 create or replace procedure printHello
1690 is
1691     begin
1692         dbms_output.put_line('hello world ');
1693     end;
1694
1695
1696 begin
1697     -- call the procedure :
1698     printHello();
1699 end;
1700 /
1701
1702 --* example 2 : update the salary of an employee
1703 create or replace procedure updateEmpSalary
1704 (id in number , eAmount in number)
1705 is
1706     expNegativeSalary exception;
1707     begin
1708
1709         if eAmount < 0 then
1710             raise expNegativeSalary;
1711         end if;
1712
1713         -- if it's ok update the salary :
1714         update employees
1715         set salary = eAmount
1716         where employee_id=id;
1717         exception
1718         when no_data_found then
1719             dbms_output.put_line('invalid id Try again ');
1720         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 updated : ');
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 updated : ');
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,

```



```

1776      which may be returned (output) as a modified value (preserve the initial value)
1777
1778  */
1779
1780  --* example 1 : out get the firstname and lastname of an employee :
1781      create or replace procedure getEmpFullname
1782      (vEmpId number ,vFirstname out employees.first_name%type, vLastname out
employees.last_name%type)
1783      is
1784      begin
1785
1786          select first_name,last_name into vFirstname,vLastname from employees
1787          where employee_id=vEmpId;
1788          exception
1789              when no_data_found then
1790                  dbms_output.put_line( chr(10) || 'invalid id [' ||vEmpId || '] try again' ||
chr(10));
1791              when others then
1792                  dbms_output.put_line('');
1793                  dbms_output.put_line('the code ' || sqlcode);
1794                  dbms_output.put_line(' tje msg ' || sqlerrm);
1795                  dbms_output.put_line('');
1796          end;
1797
1798      declare
1799          vFirstname employees.first_name%type;
1800          vLastname employees.last_name%type;
1801      begin
1802          getEmpFullname(2,vFirstname,vLastname);
1803
1804          dbms_output.put_line('the first name is : ' || vFirstname);
1805          dbms_output.put_line('the first name is : ' || vLastname);
1806      end;
1807      /
1808
1809  --* example 2 : in out : format phone
1810      create or replace procedure formatNumber
1811      (phone in out varchar)
1812      is
1813      begin
1814          phone := '(' || phone || ')';
1815      end;
1816
1817      declare
1818          vphone varchar(12) :='07715633';
1819      begin
1820          dbms_output.put_line('the phone before format : ' || vphone);
1821          formatNumber(vphone);
1822          dbms_output.put_line('the phone after format : ' || vphone);
1823
1824      end;
1825      /
1826
1827  -- default value : just for in variables
1828  -- syntax :
1829      create or replace procedure procedureName
1830      [(par1 in datatype:=defaultValue | default defaultValue )] --parameters without size :

```

```

1831  is
1832  --declare variables
1833  begin
1834  [excption]
1835
1836  end;
1837
1838  -- Available Notations forPassing Actual Parameters
1839  /*
1840  When calling a subprogram, you can write the actual
1841  parameters using the following notations:
1842  . Positional:
1843      - Lists the actual parameters in the same order as the
1844      formal parameters
1845  . Named:
1846      - Lists the actual parameters in arbitrary order and uses
1847      the association operator (=>) to associate a named formal
1848      parameter with its actual parameter
1849  . Mixed:
1850      - Lists some of the actual parameters as positional and
1851      some as named
1852  */
1853
1854  --* example :
1855  create table products
1856  (
1857  prod_id number,
1858  prod_name varchar(30),
1859  prod_type varchar(20),
1860  constraint product_pk primary key(prod_id)
1861  );
1862
1863  create or replace procedure add_product
1864  (vpro_id number , vpro_name varchar2 , vpro_type varchar2 :='sw' )
1865  is
1866  begin
1867      insert into products values
1868      (vpro_id,vpro_name,vpro_type);
1869      commit;
1870
1871      exception
1872      when others then
1873          dbms_output.put_line('error in insert ');
1874          dbms_output.put_line(sqlcode);
1875          dbms_output.put_line(sqlerrm);
1876
1877      end;
1878  --Positional
1879      execute add_product(2,'laptop');
1880  --Named
1881      execute add_product(vpro_id=>3, vpro_name=>'laptop', vpro_type=>'tech');
1882      execute add_product( vpro_name=>'laptop', vpro_id=>4 ,vpro_type=>'tech');
1883  -- Mixed
1884      execute add_product(5,vpro_name=>'laptop',vpro_type=>'tech');
1885
1886  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 /

```

```

1943     select get_sal(99) from dual;
1944
1945 -- you you want to use function with sql :
1946 /*
1947     the function should not be contains any DML (select,insert,update,delete) , commit
rollback
1948 */
1949 --* example 2 :
1950     CREATE OR REPLACE FUNCTION get_Tax(p_sal NUMBER)
1951         RETURN NUMBER
1952         AS
1953
1954     BEGIN
1955         if p_sal <5000 then
1956             return p_sal *(10/100);
1957         else
1958             return p_sal *(15/100);
1959         end if;
1960     end;
1961
1962     begin
1963         dbms_output.put_line('the tax :' || get_Tax(5000));
1964     end;
1965 /
1966     select employee_id,first_name ,get_Tax(salary)
1967     from employees;
1968
1969     select * from employees
1970     where get_Tax(salary) > 1000;
1971
1972 -- sequence :
1973 /*
1974     In database systems, a sequence is an object that generates a series of unique values.
1975     Sequences are often used to generate primary key values for tables. Here are the key
1976     aspects of sequences
1977 */
1978
1979
1980 -- create a sequence :
1981     CREATE SEQUENCE sequence_name [INCREMENT BY n | NOMAXVALUE] [START WITH n]
1982     [MAXVALUE n | NOMAXVALUE];
1983
1984 -- Retrieving Sequence Values:
1985 /*
1986     The NEXTVAL and CURRVAL functions are used to retrieve the next value
1987     and the current value
1988     of a sequence, respectively.
1989 */
1990 -- Example 1:
1991     SELECT sequence_name.NEXTVAL FROM DUAL;
1992
1993 -- reset a sequence :
1994     ALTER SEQUENCE sequence_name RESTART;
1995
1996 -- or (into the plsql program )
1997     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

```

```

2054 function functionName(par1 datatype)
2055     return datatype
2056 is
2057 begin
2058     -- code
2059 end;
2060 function functionNam2(par1 datatype)
2061     return datatype
2062 is
2063 begin
2064     -- code
2065 end;
2066 ...
2067
2068 end;
2069
2070 -- call a function from a package :
2071 packageName.functionName(arguments);
2072
2073 --* example 1 :
2074 create or replace package aria
2075 is
2076     function retangleAria
2077         ( width number , height number)
2078         return number;
2079
2080     function squareAria( side number)
2081         return number;
2082 end;
2083
2084 create or replace package body aria
2085 is
2086     function retangleAria
2087     ( width number , height number)
2088     return number
2089     as
2090     begin
2091         return width*height;
2092     exception
2093         when invalid_number then
2094             dbms_output.put_line('you should enter a number ');
2095             return null;
2096         when others then
2097             raise_application_error(sqlcode,sqlerrm);
2098             return null;
2099     end;
2100     function squareAria( side number)
2101     return number
2102     as
2103     begin
2104         return side*2;
2105     exception
2106         when invalid_number then
2107             dbms_output.put_line('you should enter a number ');
2108             return null;
2109         when others then

```

```

2110         raise_application_error(sqlcode,sqlerrm);
2111         return null;
2112     end;
2113 end;
2114
2115
2116
2117 select aria.retangleAria(5,4) from dual;
2118
2119 --* example 3 : package with variables
2120 create or replace package body aria
2121 is
2122     function retangleAria
2123     ( width number , height number)
2124     return number
2125     as
2126     begin
2127         return width*height;
2128     exception
2129         when invalid_number then
2130             dbms_output.put_line('you should enter a number ');
2131             return null;
2132         when others then
2133             raise_application_error(sqlcode,sqlerrm);
2134             return null;
2135     end;
2136     function squareAria( side number)
2137     return number
2138     as
2139     begin
2140         return side*2;
2141     exception
2142         when invalid_number then
2143             dbms_output.put_line('you should enter a number ');
2144             return null;
2145         when others then
2146             raise_application_error(sqlcode,sqlerrm);
2147             return null;
2148     end;
2149 end;
2150
2151 create or replace function get_mile_to_km
2152 (p_value number)
2153 return number
2154 is
2155     begin
2156         return global_Measurement.c_mile_to_km * p_value ;
2157     end;
2158 /
2159 create or replace function get_km_to_mille
2160 (p_value number)
2161 return number
2162 is
2163     begin
2164         return global_Measurement.c_km_to_mile * p_value ;
2165     end;

```

```

2166      /
2167
2168      create or replace package global_Measurement
2169      is
2170      c_mile_to_km constant number :=1.6093;
2171      c_km_to_mile constant number :=0.6214;
2172      end;
2173
2174      execute dbms_output.put_line('60 mile =' || get_mile_to_km (60) || 'KM');
2175      execute dbms_output.put_line('100 KM =' || get_km_to_mille(100) || 'Mille');
2176
2177      --* example 4 : declare a function in a subprogram :
2178      declare
2179          function get_sysdate
2180          return date
2181          is
2182          begin
2183              return sysdate ;
2184          end;
2185      begin
2186          dbms_output.put_line(get_sysdate);
2187      end;
2188      /
2189
2190      -- drop a package :
2191      --drop specification :
2192      drop package packageName;
2193      --drop body :
2194      drop package body packageName;
2195
2196      -- stop 53 skip until ... 86
2197
2198      -- triggers : -----:
2199      /*
2200          . A trigger is a PL/SQL block that is stored in the
2201          database and fired (executed) in response to a
2202          specified event.
2203          . The Oracle database automatically executes a trigger
2204          when specified conditions occur.
2205      */
2206      /*
2207          You can write triggers that fire whenever one of the
2208          following operations occurs in the database:
2209          . A database manipulation (DML) statement (DELETE,
2210            INSERT, or UPDATE).
2211          . A database definition (DDL) statement (CREATE, ALTER,
2212            or DROP).
2213          . A database operation such as SERVERERROR, LOGON,
2214            LOGOFF, STARTUP, or SHUTDOWN.
2215      */
2216
2217      --syntax :
2218      create or replace trigger triggerName
2219      before | after | instead of |
2220      insert | update | delete -- you combine using or
2221      on tableName

```



```

2222 begin
2223     -- code
2224     if inserting then -- check if the event is insert
2225     if deleting then -- check if the event is delete
2226     if updating then -- check if the event is update
2227     -- code
2228     end;
2229
2230 -- convert date to heour :
2231     to_number(to_char(date, 'hh24'));
2232
2233 -- extract HH:mm:ss
2234     SELECT
2235         TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) AS current_hour,
2236         TO_NUMBER(TO_CHAR(SYSDATE, 'MI')) AS current_minute,
2237         TO_NUMBER(TO_CHAR(SYSDATE, 'SS')) AS current_second
2238     FROM DUAL;
2239
2240 -- extract YYYY::MM::dd
2241     SELECT
2242         TO_CHAR(SYSDATE, 'YYYY') AS current_year,
2243         TO_CHAR(SYSDATE, 'MM') AS current_month,
2244         SELECT TO_CHAR(SYSDATE, 'DD') AS current_day FROM DUAL;
2245     FROM DUAL;
2246
2247 -- now day name :
2248     select TO_CHAR(SYSDATE, 'Day') AS day_name
2249     from dual;
2250
2251 --* example 1 :
2252     CREATE OR REPLACE TRIGGER testTrig_check_time
2253     BEFORE INSERT OR UPDATE OR DELETE
2254     ON test
2255     BEGIN
2256         IF TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) NOT BETWEEN 8 AND 13 or
TO_NUMBER(TO_CHAR(SYSDATE, 'mi'))>=30
2257         THEN
2258             RAISE_APPLICATION_ERROR(-20010, 'DML Operations not allowed now');
2259         END IF;
2260     END;
2261     /
2262
2263     BEGIN
2264         INSERT INTO test VALUES (2, 'amine', 'km', 500);
2265     END;
2266     /
2267
2268 --* example 2 : with condition predicates :
2269 -- Create the trigger
2270     CREATE OR REPLACE TRIGGER testTrig_check_time
2271     BEFORE INSERT OR UPDATE OR DELETE
2272     ON test
2273     BEGIN
2274         IF TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) NOT BETWEEN 8 AND 13 or
TO_NUMBER(TO_CHAR(SYSDATE, 'mi'))>=30
2275         THEN
2276

```

```

2277         if inserting then
2278             RAISE_APPLICATION_ERROR(-20010, 'Inserting Operation not allowed now');
2279         elsif deleting then
2280             RAISE_APPLICATION_ERROR(-20010, 'deleting Operation not allowed now');
2281         else
2282             RAISE_APPLICATION_ERROR(-20010, 'updating Operation not allowed now');
2283         end if;
2284     END IF;
2285 END;
2286 /
2287
2288 BEGIN
2289     INSERT INTO test VALUES (2, 'amine', 'km', 500);
2290
2291     delete from test;
2292 END;
2293 /
2294 begin
2295     update test
2296     set salary=500
2297     where id=1;
2298 end;
2299 /
2300 begin
2301     delete from test;
2302 end;
2303 /
2304
2305 -- Triggers row -----
2306 /*
2307     In Oracle, when working with triggers, the :NEW and :OLD qualifiers refer
2308     to the new and old values of the affected row in a table,
2309     depending on the type of trigger and the type of operation (INSERT, UPDATE, DELETE).
2310     These qualifiers are used within the body of a trigger
2311     to reference column values.
2312 */
2313 /*
2314 1-:NEW in INSERT Trigger:
2315     In the context of an INSERT trigger, :NEW represents the values that are being
2316     inserted into the table.
2317     You can reference :NEW.column_name to access the new values of specific columns.
2318
2319 2-:OLD in DELETE Trigger:
2320     In the context of a DELETE trigger, :OLD represents the values that are being
2321     deleted from the table.
2322     You can reference :OLD.column_name to access the old values of specific columns.
2323     :NEW and :OLD in UPDATE Trigger:
2324
2325 3-:NEW and :OLD in UPDATE Trigger:
2326     In the context of an UPDATE trigger, both :OLD and :NEW can be used.
2327     :OLD represents the old values before the update, and :NEW represents the new
2328     values after the update.
2329     You can compare the old and new values to perform actions based on changes.
2330 */
2331 --syntax :
2332 create or replace trigger triggerName

```

```

2333     before | after | instead of |
2334     insert | update | delete  -- you combine using or
2335     on tableName
2336     for each row
2337     begin
2338         -- code
2339         if inserting then -- check if the event is insert
2340         if deleting then -- check if the event is delete
2341         if updating then -- check if the event is update
2342         -- code
2343     end;
2344
2345 --* example 1 :
2346     CREATE OR REPLACE TRIGGER triggCheck_test_salary
2347     BEFORE
2348     UPDATE
2349     ON test
2350     for each row
2351     BEGIN
2352         if :new.salary <500 then
2353             raise_application_error(-20030,'min sal is 500');
2354             END IF;
2355     END;
2356
2357     begin
2358         update test
2359         set salary=500
2360         where id=2;
2361         commit;
2362     end;
2363     /
2364 --* example 2 :transaction history
2365     create table test_copy
2366     as select * from test;
2367     create table test_sal_audit
2368     (
2369         id number,
2370         OLD_sal number,
2371         NEW_sal number,
2372         op_date date,
2373         by_user varchar(30),
2374         description varchar2(100)
2375     );
2376
2377     --- MY VERSION GET ME AN ERROR Failed:
2378
2379     create or replace trigger trig_test_audit
2380     after
2381     insert or update or delete
2382     on test_copy
2383     for each row
2384     declare
2385         vcurrent_user VARCHAR2(30);
2386     begin
2387         select user into vcurrent_user from dual;
2388         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 trigger :
2409 alter trigger triggerName compile;
2410
2411 -- to disable all triggers on a table :
2412 alter table tableName disable all triggers;
2413
2414 -- to enable all triggers 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');

```

```

2445         end;
2446
2447         select * from customers;
2448
2449         -- triggers with view :
2450         /*
2451         1- the new record 4 will not be inserted to the
2452            original
2453            Table customer, instead of that do other
2454            transactions.
2455         2- But you can still can insert to the original table but
2456            manually using eode
2457
2458            when yo deal with view you mus update the table mainually
2459            insert
2460            delete update
2461
2462         */
2463         create or replace view vEmp_all
2464         as select * from employees;
2465
2466         create table dep_sal
2467         as
2468         select dep , sum(salary) as sumEmp
2469         from employees where dep is not null
2470         group by dep
2471         order by dep;
2472
2473         CREATE OR REPLACE TRIGGER update_dep_sal
2474         INSTEAD OF INSERT OR DELETE
2475         ON vEmp_all
2476         declare
2477         vSal number default 0;
2478         BEGIN
2479             IF INSERTING THEN
2480                 -- Insert into employees
2481                 INSERT INTO employees
2482                 VALUES (:NEW.employee_id, :NEW.first_name, :NEW.last_name, :NEW.hire_date,
2483 COALESCE(:NEW.salary, 0), :NEW.dep);
2484
2485                 -- Update dep_sal
2486                 UPDATE dep_sal
2487                 SET sumEmp = sumEmp + COALESCE(:NEW.salary, 0)
2488                 WHERE dep = :NEW.dep;
2489             ELSIF DELETING THEN
2490                 delete from employees where employee_id=:old.employee_id;
2491                 -- Update dep_sal
2492                 UPDATE dep_sal
2493                 SET sumEmp = sumEmp - COALESCE(:OLD.salary, 0)
2494                 WHERE dep = :OLD.dep;
2495             else
2496                 -- get the old salary of employee :
2497                 select salary into vSal from employees
2498                 where employee_id=:new.employee_id;
2499
2500                 --update salary

```

```

2500         update employees
2501         set salary = :new.salary
2502         where employee_id=:new.employee_id;
2503
2504     -- update dep_sal :
2505     UPDATE dep_sal
2506     SET sumEmp = sumEmp - COALESCE(:new.salary, 0) + COALESCE(vSal, 0)
2507     WHERE dep = :OLD.dep;
2508
2509     END IF;
2510 END;
2511 /
2512
2513
2514     insert into vEmp_all values (12,'km','tm',sysdate,500,1);
2515
2516     update vEmp_all
2517     set salary =1000
2518     where employee_id=12;
2519
2520     delete from vEmp_all where employee_id=12;
2521
2522     delete from vEmp_all;
2523     select * from vEmp_all;
2524     select * from dep_sal ;
2525
2526 -- stop video 93 :

```