**Day :54 19/09/22**

**CURSOR:**

* A cursor is temporary memory/SQL private area/ workspace.
  + Explicit cursors (user defined cursor)
  + Implicit cursors (system defined cursor)

**Explicit Cursors:**

* These cursors are created by user for fetching/accessing data from table in row-by-row manner.
* A cursor can store/hold multiple rows but we can access a single row only from a cursor.
* User can create a explicit cursor by the following 4 steps.
  + **Step:1 Declare A Cursor Variable**

DECLARE CURSOR <CURSOR NAME> IS <SELECT QUERY>;

* **Step:2 Open Cursor**

OPEN <CURSOR NAME>;

* **Step:3 Fetch Rows from Cursor in One by One manner**

FETCH <CURSOR NAME> INTO <VARIABLES>;

* **Step:4 Close Cursor**

CLOSE <CURSOR NAME>;

**Attributes of an Explicit Cursor:**

* They are used to check the status of a cursor.

<CURSOR NAME>% <ATTRIBUTE NAME>;

* **%open**
  + It is default attribute of a cursor.
  + It returns true when cursor connection successfully open otherwise returns false.
  + It is a Boolean type
* **%found:**
  + It returns true when cursor is having data otherwise returns false.
  + It is a Boolean type
* **%notfound:**
  + It returns true when cursor is not having data otherwise returns false
  + It is a Boolean type.
* **%rowcount:**
  + It returns no of fetch statements are executed.
  + It is a number type

Ex:

Write a cursor program to fetch a single row from a table.

SQL> DECLARE CURSOR C1 IS SELECT ENAME, SAL FROM EMP;

V\_ENAME VARCHAR2(10);

V\_SAL NUMBER(10);

BEGIN

OPEN C1;

FETCH C1 INTO V\_ENAME,V\_SAL;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME||','||V\_SAL);

CLOSE C1;

END;

/

SMITH,800

**Day :55 20/09/22**

Ex : Write a cursor program to fetch Multiple rows from a table.

SQL> DECLARE CURSOR C1 IS SELECT ENAME, SAL FROM EMP;

V\_ENAME VARCHAR2(10);

V\_SAL NUMBER(10);

BEGIN

OPEN C1;

FETCH C1 INTO V\_ENAME,V\_SAL;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME||','||V\_SAL);

FETCH C1 INTO V\_ENAME,V\_SAL;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME||','||V\_SAL);

FETCH C1 INTO V\_ENAME,V\_SAL;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME||','||V\_SAL);

CLOSE C1;

END;

/

SMITH,800

ALLEN,1600

WARD,1250

PL/SQL procedure successfully completed.

**Note:**

* Whenever we want to vetch multiple rows from a cursor instead of writing multiple times we can use looping statements like below.

1. **By using simple loop**

DECLARE CURSOR C1 IS SELECT ENAME, SAL FROM EMP;

V\_ENAME VARCHAR2(10);

V\_SAL NUMBER(10);

BEGIN

OPEN C1;

LOOP

FETCH C1 INTO V\_ENAME,V\_SAL;

EXIT WHEN C1%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME||','||V\_SAL);

END LOOP;

CLOSE C1;

END;

/

SMITH,800

ALLEN,1600

WARD,1250

JONES,2975

MARTIN,1250

BLAKE,2850

CLARK,2450

SCOTT,3000

KING,5000

TURNER,1500

ADAMS,1100

JAMES,950

FORD,3000

MILLER,1300

PL/SQL procedure successfully completed.

1. **By Using "While Loop"**

SQL> DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;

v\_ENAME VARCHAR2(10);

v\_SAL NUMBER(10);

BEGIN

OPEN C1;

FETCH C1 INTO v\_ENAME,v\_SAL;-----FETCHING STARTS FROM 1 ROW

WHILE(C1%FOUND)

LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_ENAME||' '||v\_SAL);

FETCH C1 INTO v\_ENAME,v\_SAL;-----FETCH WILL CONTINUE UPTO LAST ROW.

END LOOP;

CLOSE C1;

END;

/

SMITH 800

ALLEN 1600

WARD 1250

JONES 2975

MARTIN 1250

BLAKE 2850

CLARK 2450

SCOTT 3000

KING 5000

TURNER 1500

ADAMS 1100

JAMES 950

FORD 3000

MILLER 1300

PL/SQL procedure successfully completed.

**iii) by using "for loop"**

**=========================**

SQL> DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;

2 BEGIN

3 FOR i IN C1

4 LOOP

5 DBMS\_OUTPUT.PUT\_LINE(i.ENAME||','||i.SAL);

6 END LOOP;

7 END;

8 /

SMITH,800

ALLEN,1600

WARD,1250

JONES,2975

MARTIN,1250

BLAKE,2850

CLARK,2450

SCOTT,3000

KING,5000

TURNER,1500

ADAMS,1100

JAMES,950

FORD,3000

MILLER,1300

PL/SQL procedure successfully completed.

**NOTE:**

**=====**

- whenever we are using a "for loop" statement there is no need to write

the following statements are "open cursor,fetch from cursor and close cursor"

by explicitly(user defined) because internally oracle server will perform

these operations on cursor by automatically.

**IMPLICIT CURSORS**

* These cursors are created by system automatically when we perform DML operations over a DB table.
* Implicit cursor name is “SQL”.
* Implicit cursors are used to check the status of last DML command is executed or not.

**Attributes of Implicit cursors:**

* + **%ISOPEN:**
    - It is a default attributes which returns true when cursor connection is successfully open otherwise returns false.
  + **%FOUND:**
    - It returns true when the last DML command is executed successfully otherwise returns false.
  + **%NOTFOUND:**
    - It returns true when the last DML command is not executed successfully otherwise returns false.
  + **%ROWCOUNT:**
    - It returns the no. of rows affected with DML command.

**Ex:**

DECLARE

V\_EMPNO NUMBER(10);

BEGIN

V\_EMPNO:&V\_EMPNO;

DELETE FFROM EMP WHERE EMPNO=V\_EMPNO;

IF (SQL%FOUND) THEN

DBMS\_OUTPUT.PUT\_LINE('RECORD IS FOUND AND DELETED');

ELSE

DBMS\_OUTPUT.PUT\_LINE('RECORD IS NOT FOUND');

END IF;

END;

/

Enter value for v\_empno: 7788

RECORD IS FOUND AND DELETED

PL/SQL procedure successfully completed.

SQL> /

Enter value for v\_empno: 1122

RECORD IS NOT FOUND

PL/SQL procedure successfully completed.

**Day :56 21/09/22**

**RFL CURSOR:**

* When we assign a “select query” at the time of opening a cursor is called as “ref cursor” and it is also called as dynamic cursor.
  + Weak ref cursor
  + Strong ref Cursor

|  |  |
| --- | --- |
| **Weak Ref Cursor** | **Strong Ref Cursor** |
| It can declare without return type. | It can declare with return type. |
| It is having a pre-defined datatype for variable declaration. | There is no pre-defined datatype so that we want to create a user defined data type for strong ref cursor variable daclartion. |
| It can access data from any table (i.e. multiple table) | It can access data from a special table (single table) |

**Ex: Weak Cursor with a single table**

DECLARE

WRC SYS\_REFCURSOR;

i EMP%ROWTYPE;

BEGIN

OPEN WRC FOR SELECT \* FROM EMP WHERE DEPTNO=10;

LOOP

FETCH WRC INTO i ;

EXIT WHEN WRC%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(i.ENAME||','||i.DEPTNO);

END LOOP;

CLOSE WRC;

END;

/

MILLER,10

CLARK,10

KING,10

PL/SQL procedure successfully completed.

**Strong Cursor with a single table:**

**Syntax to create a user defined datatype**

TYPE <TYPE NAME> IS REF CURSOR RETURN TYPE<TYPE>;

**EX:**

SQL> DECLARE

TYPE UD\_REFCURSOR IS REF CURSOR RETURN EMP%ROWTYPE;

SRC UD\_REFCURSOR;

i EMP%ROWTYPE;

BEGIN

OPEN SRC FOR SELECT \* FROM EMP WHERE DEPTNO=10;

LOOP

FETCH SRC INTO i;

EXIT WHEN SRC%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(i.ENAME||','||i.DEPTNO);

END LOOP;

CLOSE SRC;

END;

/

MILLER,10

CLARK,10

KING,10

PL/SQL procedure successfully completed.

**Ex: Weak Cursor with a multiple table**

SQL> DECLARE

WRC SYS\_REFCURSOR;

i EMP%ROWTYPE;

j DEPT%ROWTYPE;

V\_DEPTNO NUMBER(10):=&V\_DEPTNO;

BEGIN

IF V\_DEPTNO =10 THEN

OPEN WRC FOR SELECT \* FROM EMP WHERE DEPTNO =10;

LOOP

FETCH WRC INTO i;

EXIT WHEN WRC%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(i.ENAME||','||i.DEPTNO);

END LOOP;

ELSIF V\_DEPTNO = 20 THEN

OPEN WRC FOR SELECT \* FROM EMP WHERE DEPTNO =20;

LOOP

FETCH WRC INTO j;

EXIT WHEN WRC%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(j.ENAME||','||j.DEPTNO);

END LOOP;

CLOSE WRC;

END IF;

END;

/

**Day :57 22/09/22**

**EXCEPTION HANDLING**

What is Exception?

* Runtime Error/ Execution Error

What is exception Handling?

* To avoid abnormal termination of a program execution.
* Pl/SQL supports the following two types of exception.
  + Pre-defined exceptions
  + User-defined exception

1. **Pre-Defined Exception:**
   1. **No Data Found:**

* If our required data is not found in a table then oracle returns an exception is called as “no data found”.

Ex:

DECLARE

V\_ENAME VARCHAR2(10);

BEGIN

SELECT ENAME INTO V\_ENAME FROM EMP WHERE

EMPNO=&EMPNO;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME);

END;

/

Enter value for empno: 7788

SCOTT

PL/SQL procedure successfully completed.

SQL> /

Enter value for empno: 1122

ERROR at line 1:

ORA-01403: no data found

ORA-06512: at line 4

* To overcome the above exception then we use a predefined exception name is “no\_data\_found ” as below,

Soulution:

DECLARE

V\_ENAME VARCHAR2(10);

BEGIN

SELECT ENAME INTO V\_ENAME FROM EMP WHERE

EMPNO=&EMPNO;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE(‘ROW DOES NOT EXIST');

END;

/

* 1. **Too\_Many\_Rows:**
* When we retrieving more than one row data from a table by using "select.....into" statement then oracle return an exception is called as "exact fetch returns more than requested number of rows".

Ex:

SQL> SELECT \* FROM TEST;

ENAME SAL

---------- ----------

A 15000

B 25000

SQL> DECLARE

v\_SAL NUMBER(10);

BEGIN

SELECT SAL INTO v\_SAL FROM TEST;

DBMS\_OUTPUT.PUT\_LINE(v\_SAL);

END;

/

ERROR at line 1:

ORA-01422: exact fetch returns more than requested number of rows

ORA-06512: at line 4

- to overcome the above exception then we use a pre-defined

exception name is "too\_many\_rows".

**Solution:**

DECLARE

v\_SAL NUMBER(10);

BEGIN

SELECT SAL INTO v\_SAL FROM TEST;

DBMS\_OUTPUT.PUT\_LINE(v\_SAL);

EXCEPTION

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('A TABLE IS HAVING MORE THAN ONE ROW');

END;

/

A TABLE IS HAVING MORE THAN ONE ROW

PL/SQL procedure successfully completed.

* 1. **Zero\_Divide:**
* when we perform a division with zero then oracle return an exception is called as "divisor is equal to zero".

Ex:

SQL> DECLARE

2 X NUMBER(10);

3 Y NUMBER(10);

4 Z NUMBER(10);

5 BEGIN

6 X:=&X;

7 Y:=&Y;

8 Z:=X/Y;

9 DBMS\_OUTPUT.PUT\_LINE(Z);

10 END;

11 /

Enter value for x: 10

Enter value for y: 2

5

PL/SQL procedure successfully completed.

SQL> /

Enter value for x: 10

Enter value for y: 0

ERROR at line 1:

ORA-01476: divisor is equal to zero

ORA-06512: at line 8

- to overcome the above an exception then we use oracle

pre-defined exception name is "zero\_divide".

**Solution:**

DECLARE

X NUMBER(10);

Y NUMBER(10);

Z NUMBER(10);

BEGIN

X:=&X;

Y:=&Y;

Z:=X/Y;

DBMS\_OUTPUT.PUT\_LINE(Z);

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('SECOND NUMBER SHOULD NOT BE ZERO');

END;

Enter value for x: 10

Enter value for y: 5

2

PL/SQL procedure successfully completed.

SQL> /

Enter value for x: 10

Enter value for y: 0

SECOND NUMBER SHOULD NOT BE ZERO

PL/SQL procedure successfully completed.

* 1. **SQLCODE & SQLERRM:**
* These are pre-defined properties which are used to handling any type of exception which was raised in a pl/sql block.
* When we use these two properties then we must use "others" exception name.
* Here,

SQLCODE : is return exception number.

SQLERRM : is return exception message.

Ex:

DECLARE

X NUMBER(10);

Y NUMBER(10);

Z NUMBER(10);

BEGIN

X:=&X;

Y:=&Y;

Z:=X/Y;

DBMS\_OUTPUT.PUT\_LINE(Z);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLCODE);

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

Enter value for x: 10

Enter value for y: 5

2

PL/SQL procedure successfully completed.

SQL> /

Enter value for x: 10

Enter value for y: 0

-1476

ORA-01476: divisor is equal to zero

PL/SQL procedure successfully completed.

1. **User-Defined Exception:**

* These exceptions arebcreated by user to raise and handling exceptions in PL/SQL block.
* When we create our own exception names then we follow the following 3 steps;
  + Step1: Declare user defined exception

<UD EXCEPTION NAME> EXCEPTION;

* Step2: To raise a UD exception name
* Method:1

RAISE <UD DEFINED EXCEPTION NAME>;

* Method: 2

RAISE\_APPLICATION\_ERROR (NUMBER, MESSAGE);

* “raise” statement can raise and also handling an exception whereas “raise\_application\_error()” can raise an exception but not handling exception.
* Step3: Handling exceptions with UD exception name

EXCEPTION

WHEN <UD EXCEPTION NAME> THEN

<HANDLING EXCEPTION STATEMENT>;

END;

/

**Day :58 23/09/22**

EX:

SQL> DECLARE

X NUMBER(10);

Y NUMBER(10);

Z NUMBER(10);

EX EXCEPTION;

BEGIN

X:=&X;

Y:=&Y;

IF Y=0 THEN

RAISE EX;

ELSE

Z:=X/Y;

DBMS\_OUTPUT.PUT\_LINE(Z);

END IF;

EXCEPTION

WHEN EX THEN

DBMS\_OUTPUT.PUT\_LINE('SECOND NUMBER NOT BE ZERO');

END;

/

Enter value for x: 10

Enter value for y: 5

2

PL/SQL procedure successfully completed.

SQL> /

Enter value for x: 10

Enter value for y: 0

SECOND NUMBER NOT BE ZERO

PL/SQL procedure successfully completed.

raise\_application\_error(number,message):

* It is a pre-defined method which is used to raise an exception but not handl exception in pl/sql block.
* It can raise an exception in oracle server format.
* It is having two arguments are:

number: it returns an exception user defined number.

it should be -20001 to -20999

message: it returns user defined error message.

SQL> DECLARE

X NUMBER(10);

Y NUMBER(10);

Z NUMBER(10);

EX EXCEPTION;

BEGIN

X:=&X;

Y:=&Y;

IF Y=0 THEN

RAISE EX;

ELSE

Z:=X/Y;

DBMS\_OUTPUT.PUT\_LINE(Z);

END IF;

EXCEPTION

WHEN EX THEN

RAISE\_APPLICATION\_ERROR(-20478,'SECOND NUMBER NOT BE ZERO');

END;

/

Enter value for x: 10

Enter value for y: 5

2

PL/SQL procedure successfully completed.

SQL> /

Enter value for x: 10

Enter value for y: 0

ERROR at line 1:

ORA-20478: SECOND NUMBER NOT BE ZERO

ORA-06512: at line 17

**EXCEPTION PROPAGATION:**

* Generally, exceptions are raised in execution block and those exceptions are handling with pre-defined exeption name / user defined exception name by default.
* But when an exception is raised in declaration block then we use an exception propagation mechanism to handle exceptions which raised in declaration block.
* When we use exception propagation then we implement nested pl/sql block.

Ex:

SQL> DECLARE

2 X VARCHAR2(3):='KING';

3 BEGIN

4 DBMS\_OUTPUT.PUT\_LINE(X);

5 EXCEPTION

6 WHEN VALUE\_ERROR THEN

7 DBMS\_OUTPUT.PUT\_LINE('INVALID STRING LENGTH');

8 END;

9 /

ERROR at line 1:

ORA-06502: PL/SQL: numeric or value error: character string buffer too small

ORA-06512: at line 2

* To handle the above an exception then we use a mechanism

is called as "exception propagation".

**Solution:**

BEGIN

DECLARE

X VARCHAR2(3):='KING';

BEGIN

DBMS\_OUTPUT.PUT\_LINE(X);

EXCEPTION

WHEN VALUE\_ERROR THEN

DBMS\_OUTPUT.PUT\_LINE('INNER BLOCK CAN HANDLED');

END;

EXCEPTION

WHEN VALUE\_ERROR THEN

DBMS\_OUTPUT.PUT\_LINE('OUTER BLOCK CAN HANDLED');

END;

/

OUTER BLOCK CAN HANDLED

PL/SQL procedure successfully completed.

**Note:**

* In exception propagation mechanism declaration block

exceptions are handling by outer block only.

**Day :59 24/09/22**

**Sub Blocks**

* It is a named block which can store that we can written under subblock.
* Four Types:

1. Stored Procedure
2. Stored Function
3. Package
4. Trigger

**Stored Procedure:**

* It is a named block which can accept some input values from the user and to perform some operation and it may be/ may not be return a value.
* If we pass “out” parameters to a stored procedure then it returns a value other procedure will never return a value.
* Syntax:

CREATE [OR REPLACE] PROCEDURE<PNAME> (<PARAMETER NAME> [MODE TYPE] <DATA TYPE>) IS /AS

<DECLARE VARIABLES>;

BEGIN

<PROCEDURE BODY/ STATEMENTS>;

END;

/

How to call a stored Procedure:

Syntax:1

EXECUTE <PNAME> (VALUE/(S));

(or)

EXEC <PNAME> (VALUE/(S));

Syntax:2 Calling with Anonymous Block

BEGIN

<PNAME> (VALUE/(S));

END;

/

**Types of parameters modes:**

* In PL/SQL stored procedure is supporting the following 3 types of parameters modes.
  + IN
  + OUT
  + IN OUT

IN:

* These default parameters of a stored procedure.
* To store input values which was given by user at the time execution of a stored procedure.

OUT:

* Generally, stored procedures will never return a value if stored procedure want to return a value, then we use “out’ parameters.

IN OUT:

* This parameter can accept and also return a value

**Examples of “IN” parameters:**

* 1. Create a stored procedure to accept empno as a “in” parameter and display that employee ename and salary details from emp table?

CREATE OR REPLACE PROCEDURE SP1(P\_EMPNO in NUMBER)

IS

V\_ENAME VARCHAR2(10);

V\_SAL NUMBER(10);

BEGIN

SELECT ENAME, SAL INTO V\_ENAME, V\_SAL FROM EMP

WHERE EMPNO=P\_EMPNO;

DBMS\_OUTPUT.PUT\_LINE(V\_ENAME||','||V\_SAL);

END;

/

Syntax:1

SQL> EXECUTE SP1(7788);

SCOTT,3000

PL/SQL procedure successfully completed.

Syntax:2

SQL> BEGIN

2 SP1(7566);

3 END;

4 /

JONES,2975

Note:

If we want to view subblock objects (SP/SF/package/tigger)

Ex:

SQL> DESC USER\_OBJECTS;

SQL> SELECT OBJECT\_NAME FORM USER\_OBJECTA WHERE OBJECT\_TYPE = ‘PROCEDURE”;

Note:

* If we want to view the source code of a particular subblock object (sp/sf/package/tigger) then we use a datadictionary is called as “user\_source”.

Ex:

SQL> DESC USER\_SOURCE;

SQL> SELECT TEXT FROM USER\_SOURCE WHERE NAME = ‘SP1’;

**Examples of “OUT” parameters:**

CREATE OR REPLACE PROCEDURE SP2(X IN NUMBER, Y OUT NUMBER)

AS

BEGIN

Y:=X\*X;

END;

/

Procedure created.

SQL> EXECUTE SP2(5); 🡺 Error: wrong number or types of arguments in call to 'SP2'

**Day :60 26/09/22**

* To overcome the above problem, we need to follow the

following three steps are

Step1: declare binding variable (referenced variable) for "out" parameters:

Syntax:

var[iable] <bind variable name> <datatype>[size];

Step2: adding bind variables to a stored procedure:

Syntax:

execute <pname>(value1,value2,....,:<bind variable name1>,...);

Step3: print bind variables:

Syntax:

print <bind variable name>;

OUTPUT:

SQL> VAR A NUMBER;

SQL> EXECUTE SP2(5,:A);

PL/SQL procedure successfully completed.

SQL> PRINT A;

A

----------

25

EX:

Create a SP to input empno as a "IN" parameter and return that employee provident fund and professional tax at 10%,5% on basic salary by using "OUT" parameters?

SQL> CREATE OR REPLACE PROCEDURE SP3(p\_EMPNO IN NUMBER,PF OUT NUMBER,PT OUT NUMBER)

2 IS

3 v\_BSAL NUMBER(10);

4 BEGIN

5 SELECT SAL INTO v\_BSAL FROM EMP WHERE EMPNO=p\_EMPNO;

6 PF:=v\_BSAL\*0.1;

7 PT:=v\_BSAL\*0.05;

8 END;

9 /

Procedure created.

SQL> VAR bPF NUMBER;

SQL> VAR bPT NUMBER;

SQL> EXECUTE SP3(7788,:bPF,:bPT);

PL/SQL procedure successfully completed.

SQL> PRINT bPF bPT;

BPF

----------

300

BPT

----------

150

**Example ON "IN OUT":**

EX:

SQL> CREATE OR REPLACE PROCEDURE SP4(X IN OUT NUMBER)

2 IS

3 BEGIN

4 X:=X\*X\*X;

5 END;

6 /

Procedure created.

SQL> EXECUTE SP4(5);

ERROR at line 1:

ORA-06550: line 1, column 11:

PLS-00363: expression '5' cannot be used as an assignment

target.

* To overcome the above problem we should follow the

following four steps are;

Step1: declare binding variable(referenced variable):

Syntax:

var[iable] <bind variable name> <datatype>[size];

Step2: to assign a value to a bind varibale:

Syntax:

execute :<bind variable name> := <value>;

Step3: adding bind variables to a stored procedure:

Syntax:

execute <pname>(:<bind variable name1>);

Step4: print bind variables:

Syntax:

print <bind variable name>;

OUTPUT:

SQL> VAR A NUMBER;

SQL> EXECUTE :A:=5;

PL/SQL procedure successfully completed.

SQL> EXECUTE SP4(:A);

PL/SQL procedure successfully completed.

SQL> PRINT A;

A

----------

125

**Syntax To Drop A Stored Procedure:**

DROP PROCEDURE <PNAME>;

EX: DROP PROCEDURE SP1;

**Day :61 27/09/22**

**STORED FUNCTIONS**

* A function is block of code to perform some tasks and must return a value.
* These functions are created by User explicitly.
* So that we can also called as “user Defined function”.
* Syntax:

CREATE OR REPLACE FUNCTION <FUNCTION\_NAME>

[(ARUGMENT DATATYPE,

ARGUMENT DATATYPE,)]

RETURN <DATATYPE>

IS

BEGIN

<EXEC-STATEMENTS>;

RETURN (VALUE);

END <FUNCTION\_NAME>;

/

**How To Call a Stored Function:**

SELECT <FNAME>(VALUES) FROM DUAL;

EX:

Create a SF to return the no.of employees are joined in between the given two dates?

SQL> CREATE OR REPLACE FUNCTION SF3(SD DATE,ED DATE)

2 RETURN NUMBER

3 AS

4 v\_NOEMP NUMBER(10);

5 BEGIN

6 SELECT COUNT(\*) INTO v\_NOEMP FROM EMP

7 WHERE HIREDATE BETWEEN SD AND ED;

8 RETURN v\_NOEMP;

9 END;

10 /

Function created.

SQL> SELECT SF3(‘01-JAN-81’,’31-DEC-81’) FROM DUAL;

SF3(‘01-JAN-81’,’31-DEC-81’)

10

EX:

Create a SF to accept employee number and return that employee gross salary based on the following conditions are

i)HRA -------- 10%

ii) DA -------- 20%

iii) PF ------- 10%.

SQL> CREATE OR REPLACE FUNCTION SF4(p\_EMPNO NUMBER)

2 RETURN NUMBER

3 AS

4 v\_BSAL NUMBER(10);

5 v\_HRA NUMBER(10);

6 v\_DA NUMBER(10);

7 v\_PF NUMBER(10);

8 v\_GROSS NUMBER(10);

9 BEGIN

10 SELECT SAL INTO v\_BSAL FROM EMP WHERE EMPNO=p\_EMPNO;

11 v\_HRA := v\_BSAL\*0.1;

12 v\_DA := v\_BSAL\*0.2;

13 v\_PF := v\_BSAL\*0.1;

14 v\_GROSS := v\_BSAL+v\_HRA+v\_DA+v\_PF;

15 RETURN v\_GROSS;

16 END;

17 /

Function created.

SQL> SELECT SF4(7788) FROM DUAL;

SF4(7788)

4200

EX:

SQL> SELECT OBJECT\_NAME FROM USER\_OBJECTS

WHERE OBJECT\_TYPE=’FUNCTION’;

SQL> SELECT TEXT FROM USER\_SOURCE WHERE NAME=’SF1’;

**Syntax To Drop A Function:**

SQL> DROP FUNCTION SF1;

**Package**

* It is collection of variables, stored procedures and stored functions are in a single unit of memory.
* To improve the performance of accessing data from an application.
* Packages are also providing reusability of code and supporting to implementing function overloading mechanism in oracle database.
* To create a package we need to follow the following two blocks those are,
  1. Package specification block
  2. Package implementation block(body)

**Package specification block:**

* In this block we can declare variables, stored procedures and stored functions.
* Syntax:

CREATE [OR REPLACE] PACKAGE <PNAME>

IS/AS

<DECLARE VARIABLES>;

<DECLARE STORED PROCEDURES>;

<DECLARE STORED FUNCTIONS>;

END;

/

**Package Implementation Block (Body):**

* In this block we can implementing logical code of store procedures and stored functions which was declared in package specification block.
* Syntax:

CREATE [OR REPLACE] PACKAGE BODY <PNAME>

IS/AS

<IMPLEMENTING PROCEDURE LOGICAL CODE>

<IMPLEMENTING UNCTION LOGICAL CODE>;

END;

/

**How to call a stored procedure from a package:**

EXECUTE <PNAME>.<PROCEDURE NAME>(VALUE/(S));

**How to call a stored function from a package:**

SELECT <PNAME>.<FNAME>(VALUE/(S)) FROM DUAL;

EX:

Create a package to bind multiple stored procedures?

SQL> CREATE OR REPLACE PACKAGE PK1

2 IS

3 PROCEDURE SP1;

4 PROCEDURE SP2;

5 END;

6 /

Package created.

SQL> CREATE OR REPLACE PACKAGE BODY PK1

2 IS

3 PROCEDURE SP1

4 AS

5 BEGIN

6 DBMS\_OUTPUT.PUT\_LINE(‘I AM PROCEDURE1’);

7 END SP1;

8 PROCEDURE SP2

9 AS

10 BEGIN

11 DBMS\_OUTPUT.PUT\_LINE(‘I AM PROCEDURE2’);

12 END SP2;

13 END;

14 /

Package body created.

SQL> EXECUTE PK1.SP1;

I AM PROCEDURE1

PL/SQL procedure successfully completed.

SQL> EXECUTE PK1.SP2;

I AM PROCEDURE2

PL/SQL procedure successfully completed.

EX:

create a package to bind stored procedure and stored function

along with a variable?

SQL> CREATE OR REPLACE PACKAGE PK2

2 IS

3 X NUMBER(10):=5000;

4 PROCEDURE SP1;

5 FUNCTION SF1(A NUMBER) RETURN NUMBER;

6 END;

7 /

Package created.

SQL> CREATE OR REPLACE PACKAGE BODY PK2

2 IS

3 PROCEDURE SP1

4 AS

5 A NUMBER(10);

6 BEGIN

7 A:=X/2;

8 DBMS\_OUTPUT.PUT\_LINE(A);

9 END SP1;

10 FUNCTION SF1(A NUMBER)

11 RETURN NUMBER

12 AS

13 BEGIN

14 RETURN X\*A;

15 END SF1;

16 END;

17 /

Package body created.

SQL> EXECUTE PK2.SP1;

2500

PL/SQL procedure successfully completed.

SQL> SELECT PK2.SF1(2) FROM DUAL;

PK2.SF1(2)

10000

**How To Implement Function Overloading By Using Package:**

* When no.of arguments are changed to the same function then we called as function overloading.

Ex:

function sf1(x number,y number);

function sf1(a number,b number,c number);

**Day :62 28/09/22**

**TRIGGERS**

* It is a named block similar to stored procedure but executed (called) by system automatically.
* **Purpose of Triggers:**
  + To raise alerts along with security.
  + To control/ restrict DDL, DML operations based on some business logical conditions.
  + To validate Data
  + For Auditing.
* **Types of Triggers:**
  + 1. DML Triggers
    2. DDL Triggers (DB Triggers)

**DML Triggers:**

* When we create a trigger object based on DML commands (Insert, update, delete) then we called as “DML Trigger”.
* These triggers are executed by system automatically when we perform DML operations on a particular table.
* Syntax:

CREATE [OR REPALCE] TRIGGER <TRIGGER NAME>

BEFORE/ AFTER INSERT(OR) UPDATE (OR) DELETE ON <TABLE NAME/OBJECT NAME>

[FOR EACH ROW] 🡺used in row level triggers only.

BEGIN

<TRIGGER BODY/ STATEMENT>;

END;

/

* **Before Event:**
  + When we create a trigger with “before event” then,

First: Execute Trigger body

Later: Execute DML Command.

* **After Event:**
  + - When we create a trigger with “After event” then,

First: Execute DML Command.

Later: Execute Trigger body

* **Note:**
  + Both events triggers are providing same results.

**Levels of Triggers:**

* Trigger can be created at two levels
  + Row level Triggers
  + Statement level Triggers

Row Level Triggers:

* In this level a trigger is executing for each row wise in a table.
* So that we should use a clause statement is “for each row”.

Ex:

SQL> SELECT \* FROM TEST;

ENAME SAL

---------- ----------

SMITH 15000

ALLEN 25000

WARD 15000

JONES 31000

EX:

SQL> CREATE OR REPLACE TRIGGER TR1

2 AFTER UPDATE ON TEST

3 FOR EACH ROW

4 BEGIN

5 DBMS\_OUTPUT.PUT\_LINE('HELLO');

6 END;

7 /

Trigger created.

TESTING:

SQL> UPDATE TEST SET SAL=12000 WHERE SAL=15000;

HELLO

HELLO

2 rows updated.

**Day :63 29/09/22**

**Statement level triggers:**

* In this level a trigger body is executing only one time for a DML operation.

EX:

SQL> CREATE OR REPLACE TRIGGER TR1

2 AFTER UPDATE ON TEST

3 BEGIN

4 DBMS\_OUTPUT.PUT\_LINE('HAI');

5 END;

6 /

Trigger created.

SQL> UPDATE TEST SET SAL=8000 WHERE SAL=12000;

HAI

2 rows updated.

**Bind Variables:**

* These are just like normal variables which are used to store the value while inserting, updating and deleting data from a table.
* There are two types of binding variables:
  + :NEW
    - When we insert new values into a table those new values are storing under :new bind variable.
    - Syntax:

:new.<column name> = <value>

* :OLD
  + When we delete old values from a table those old values are storing under :old bind variable.
  + Syntax:

:old.<column name> = <value>

**Note:**

* These bind Variables are used in row level triggers only.

**Example to raise alerts along with security:**

CREATE OR REPLACE TRIGGER TRALERT

AFTER INSERT ON TEST

BEGIN

RAISE\_APPLICATION\_ERROR (-20458,'SOMEONE IS INSERTING DATA IN YOUR TEST TABLE ');

END;

/

Testing:

SQL> INSERT INTO TEST VALUES('AARA',9000);

INSERT INTO TEST VALUES('AARA',9000)

\*

ERROR at line 1:

ORA-20458: SOMEONE IS INSERTING DATA IN YOUR TEST TABLE

ORA-06512: at "DEEPI.TRALERT", line 2

ORA-04088: error during execution of trigger 'DEEPI.TRALERT'

**Ex:**

CREATE OR REPLACE TRIGGER TRALERT

AFTER INSERT OR UPDATE OR DELETE ON TEST

BEGIN

RAISE\_APPLICATION\_ERROR (-20458,'SOMEONE IS INSERTING DATA IN YOUR TEST TABLE ');

END;

/

Testing:

SQL> DELETE FROM TEST WHERE ENAME='DEEPI';

DELETE FROM TEST WHERE ENAME='DEEPI'

\*

ERROR at line 1:

ORA-20458: SOMEONE IS INSERTING DATA IN YOUR TEST TABLE

ORA-06512: at "DEEPI.TRALERT", line 2

ORA-04088: error during execution of trigger 'DEEPI.TRALERT'

EX:

CREATE OR REPLACE TRIGGER TRALERT

AFTER INSERT OR UPDATE OR DELETE ON TEST

BEGIN

RAISE\_APPLICATION\_ERROR(-20458,'SOME ONE PERFORMING DML OPERATION ON YOUR TABLE.PLZ CHECK!!!');

END;

/

TESTING:

INSERT INTO TEST VALUES('MILLER',45000);

UPDATE TEST SET SAL=12000 WHERE SAL=8000'

DELETE FROM TEST WHERE ENAME='WARD';

ERROR at line 1:

ORA-20458: SOME ONE PERFORMING DML OPERATION ON YOUR TABLE.PLZ CHECK!!!

EX.on control / restricted dml operations based on some business logical conditions:

create a trigger to restricted all DML operations on a table on every thursday?

SQL> CREATE OR REPLACE TRIGGER TRDAY

2 BEFORE INSERT OR UPDATE OR DELETE ON TEST

3 BEGIN

4 IF TO\_CHAR(SYSDATE,'DY') = 'THU' THEN

5 RAISE\_APPLICATION\_ERROR(-20147,'WE CANNOT PERFORM DML OPERATIONS ON TEST TABLE

6 ON THURSDAY');

7 END IF;

8 END;

9 /

Trigger created.

TESTING

SQL> DELETE FROM TEST WHERE ENAME='WARD';

ERROR at line 1:

ORA-20147: WE CANNOT PERFORM DML OPERATIONS ON TEST TABLE ON THURSDAY

EX:

create a trigger to restricted all DML operations on a table in between

9am to 5pm ?

SQL> CREATE OR REPLACE TRIGGER TRTIME

2 AFTER INSERT OR UPDATE OR DELETE ON TEST

3 BEGIN

4 IF TO\_CHAR(SYSDATE,'HH24') BETWEEN 9 AND 16 THEN

5 RAISE\_APPLICATION\_ERROR(-20584,'INVALID TIME');

6 END IF;

7 END;

8 /

TESTING

SQL> DELETE FROM TEST WHERE ENAME='SMITH';

ERROR at line 1:

ORA-20584: INVALID TIME

LOGIC:

======

9AM(9) = 9:00:00AM TO 9:59:59AM------COMES UNDER 9 O CLOCK

5PM(17) = 5:00:00PM TO 5:59:59PM-----UPTO 6 O CLOCK

4PM(16) = 4:00:00PM TO 4:59:59PM -----UPTO 5 O CLOCK