**EXPERIMENT – 11**

**AIM :**

Demonstrate the concept of Exception Handling**.**

**THEORY :**

In this experiment we will learn about Exception Handling. An exception is an error which disrupts the normal flow of program instructions. PL/SQL provides us the exception block which raises the exception thus helping the programmer to find out the fault and resolve it.

There are two types of exceptions defined in PL/SQL:

1. **SYSTEM DEFINED EXCEPTIONS –** The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).
2. **USER DEFINED EXCEPTION –** The sequence-structure simply executes a sequence of statements in the order in which they occur.

**SYNTAX:**

WHEN exception THEN

Statement;

DECLARE

declarations section;

BEGIN

executable command(s);

EXCEPTION

WHEN exception1 THEN

statement1;

WHEN exception2 THEN

statement2;

[WHEN others THEN]

/\* default exception handling code \*/

END;

**1. SYSTEM DEFINED EXCEPTIONS**

These exceptions are predefined in PL/SQL which get raised WHEN certain database rule is violated.

System-defined exceptions are further divided into two categories:

(i) Named system exceptions.

(ii) Unnamed system exceptions.

**(i) NAMED SYSTEM EXCEPTIONS**

They have a predefined name by the system like ACCESS\_INTO\_NULL, DUP\_VAL\_ON\_INDEX, LOGIN\_DENIED etc. the list is quite big.

**(a) NO\_DATA\_FOUND :** It is raised WHEN a SELECT INTO statement returns no rows.

For eg:

DECLARE

temp varchar(20);

BEGIN

SELECT g\_id into temp from Student where g\_name = 'Ramesh';

exception

WHEN no\_data\_found THEN

dbms\_output.put\_line('ERROR');

dbms\_output.put\_line('There is no name as');

dbms\_output.put\_line(' Ramesh in geeks table');

end;

**OUTPUT:**

ERROR

There is no name as Ramesh in Student table

**(b)** **TOO\_MANY\_ROWS :** It is raised WHEN a SELECT INTO statement returns more than one row.

For eg:

DECLARE

temp varchar(20);

BEGIN

-- raises an exception as SELECT

-- into trying to return too many rows

SELECT g\_name into temp from STUDENT;

dbms\_output.put\_line(temp);

EXCEPTION

WHEN too\_many\_rows THEN

dbms\_output.put\_line('Error trying to SELECT too many rows');

end;

**OUTPUT:**

Error trying to SELECT too many rows

**(c) VALUE\_ERROR :** This error is raised WHEN a statement is executed that resulted in an arithmetic, numeric, string, conversion, or constraint error. This error mainly results from programmer error or invalid data input.

For eg:

DECLARE

temp number;

BEGIN

SELECT g\_name into temp from STUDENT where g\_name = 'Suraj';

dbms\_output.put\_line('The g\_name is '||temp);

EXCEPTION

WHEN value\_error THEN

dbms\_output.put\_line('Error');

dbms\_output.put\_line('Change data type of temp to varchar(20)');

END;

**OUTPUT:**

Error

Change data type of temp to varchar(20)

**(d) ZERO\_DIVIDE :** Raises exception WHEN dividing with zero.

For eg:

DECLARE

a int:=10;

b int:=0;

answer int;

BEGIN

answer:=a/b;

dbms\_output.put\_line('the result after division is'||answer);

exception

WHEN zero\_divide THEN

dbms\_output.put\_line('Dividing by zero please check the values again');

dbms\_output.put\_line('The value of a is '||a);

dbms\_output.put\_line('The value of b is '||b);

END;

**OUTPUT:**

Dividing by zero please check the values again

The value of a is 10

The value of b is 0

**(ii) UNNAMED SYSTEM EXCEPTIONS**

Oracle doesn’t provide name for some system exceptions called unnamed system exceptions.These exceptions don’t occur frequently.These exceptions have two parts code and an associated message.

The way to handle to these exceptions is to assign name to them using Pragma EXCEPTION\_INIT.

**SYNTAX:**

PRAGMA EXCEPTION\_INIT(exception\_name, -error\_number);

Error\_number are pre-defined and have negative integer range from -20000 to -20999.

For eg:

DECLARE

exp exception;

pragma exception\_init (exp, -20015);

n int:=10;

BEGIN

FOR i IN 1..n LOOP

dbms\_output.put\_line(i\*i);

IF i\*I = 9 THEN

RAISE exp;

END IF;

END LOOP;

EXCEPTION

WHEN exp THEN

dbms\_output.put\_line('HELLO WORLD');

END;

**OUTPUT:**

1

4

9

HELLO WORLD

**2. USER DEFINED EXCEPTIONS**

This type of users can create their own exceptions according to the need and to raise these exceptions explicitly raise command is used.

For eg:

Divide non-negative integer x by y such that the result is greater than or equal to 1.

From the given question we can conclude that there exist two exceptions

* Division be zero
* If result is greater than or equal to 1 means y is less than or equal to x.

DECLARE

x int: = &x; /\*taking value at run time\*/

y int: = &y;

div\_r float;

exp1 EXCEPTION;

exp2 EXCEPTION;

BEGIN

IF y=0 then

raise exp1;

ELSEIF y > x then

raise exp2;

ELSE

div\_r:= x / y;

dbms\_output.put\_line('The result is '||div\_r);

END IF;

EXCEPTION

WHEN exp1 THEN

dbms\_output.put\_line('Error');

dbms\_output.put\_line('Division by zero not allowed');

WHEN exp2 THEN

dbms\_output.put\_line('Error');

dbms\_output.put\_line('y is greater than x please check the input');

END;

**INPUT 1**: x = 20

y = 10

**OUTPUT:** The result is 2

**INPUT 2:** x = 20

y = 0

**OUTPUT:** Error

Division by zero not allowed

**INPUT 3:** x=20

y = 30

**OUTPUT:** Error

y is greater than x please check the input

**RAISE\_APPLICATION\_ERROR**:

It is used to display user - defined error messages with error number whose range is in between -20000 and -20999. When RAISE\_APPLICATION\_ERROR executes it returns error message and error code which looks same as Oracle built-in error.

For eg:

DECLARE

myex EXCEPTION;

n NUMBER :=10;

BEGIN

FOR i IN 1..n LOOP

dbms\_output.put\_line(i\*i);

IF i\*i=36 THEN

RAISE myex;

END IF;

END LOOP;

EXCEPTION

WHEN myex THEN

RAISE\_APPLICATION\_ERROR(-20015, 'HELLO WORLD’);

END;

**OUTPUT:**

Error report:

ORA-20015: HELLO WORLD

ORA-06512: at line 13

1

4

9

16

25

36